Enrico Gregorio parla di LaTeX3

Gianluca Pignalberi, Enrico Gregorio and Frank Mittelbach

Sommario Enrico Gregorio (con uno *spin-off* inatteso), intervistato in merito, ci parla del presente e del futuro di LaTeX3, analizzandone col codice alla mano vantaggi e potenzialità.

Abstract Enrico Gregorio (with the help of an unexpected spin-off), interviewed about it, tells us about the present and the future of LaTeX3, analyzing advantages and potential with meaningful code snippets.

Enrico Gregorio non ha bisogno di presentazioni. Potrebbe tranquillamente essere il condiscendente utente Unix ritratto da Scott Adams in Dilbert (figura 1) nell'aspetto e, soprattutto, nella conoscenza, se non fosse che non indossa *mai* bretelle. Al 31 gennaio 2022, il numero delle sue risposte sulla sola piattaforma StackExchange su TeX e derivati supera 22 000, con una reputazione di oltre 960 000. Il numero di pacchetti da lui scritti (in base a CTAN) è 17 (senza contare quelli in cui è intervenuto senza prendersene il merito come il "mio" ormai obsoleto combelow).

Al GUIT *meeting 2020* qualche cassandra ha fatto delle osservazioni, a mio parere ingenerose, sul futuro di LaTeX ed ho voluto capirne di più con l'aiuto di chi LaTeX lo vive e quotidianamente lo mantiene e migliora in tantissimi modi.

GIANLUCA PIGNALBERI: Enrico, potresti riassumere la storia di TeX in dieci date rappresentative?

ENRICO GREGORIO: Dieci date che giudico significative sono:

1978 prima versione di TeX

1982 seconda versione di TeX, con molti cambiamenti

1984 prima versione di LaTeX

1985 terza versione di TeX, con alcuni cambiamenti

1989 AMS-TeX

1991 NFSS

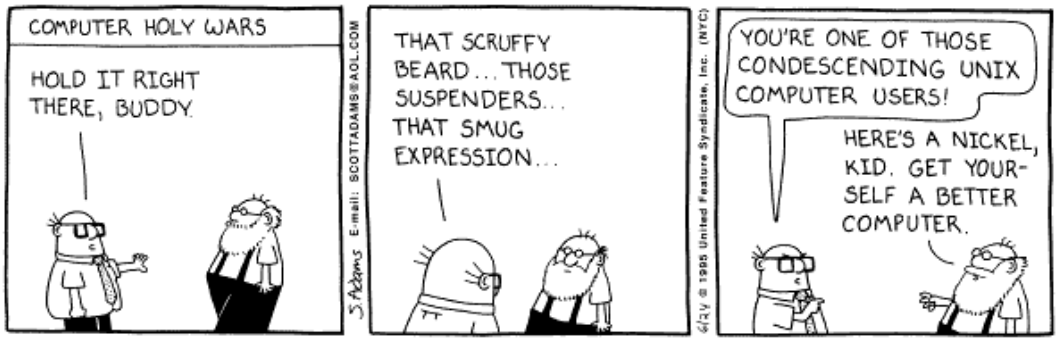

Figura 1. Famosissima striscia di Dilbert sull'archetipico utente Unix.

1992 $\mathcal{AMS}$-LaTeX

1993 teTeX

1994 LaTeX 2_ε

1996 TeX Live

G.P.: Da molti anni "un fantasma si aggira per il mondo". No, non è il topo di DeLillo che cita Marx e il comunismo in Europa: è LaTeX3. È significativo il fatto che tu non lo abbia incluso nel precedente elenco? Ce ne parli?

E.G.: Lo sviluppo di expl3 è cominciato subito dopo il rilascio di LaTeX 2_ε. Le prime idee risalgono al 1995, con la distinzione tra funzioni e variabili. Il problema, per parecchi anni, è stato la difficoltà di ottenere un kernel funzionante date le restrizioni di memoria. Perciò è rimasto silente fino a che Joseph Wright notò le potenzialità del linguaggio e lo impiegò per scrivere la seconda versione di siunitx nel 2010.
Il che causò il suo ingresso nel LaTeX team. Poco dopo venne cooptato anche Bruno Le Floch, e lo sviluppo fu piuttosto tumultuoso. Nel 2012 ci entrai anch'io dopo l'uscita di kantlipsum, un fondamentale pacchetto di cui tutti sentivano il bisogno. Scherzi a parte, cominciai ad adoperarlo per varie risposte su TeX.SE ma anche per xpatch, cominciando anche a sviluppare regexpatch: un buon numero di funzionalità di l3regex sono il risultato della corrispondenza per risolvere alcuni problemi.
Molte parti di expl3 sono state ammodernate, alcune abbandonate, altre modificate. Dalla versione 2020-10-01, expl3 è entrato nel kernel di LaTeX.
Fino a un paio d'anni fa, l'idea era di sviluppare LaTeX3, ma non accadrà mai: un nuovo kernel avrebbe mandato in soffitta migliaia di pacchetti e nessuno l'avrebbe usato. Quindi la strategia è di inserire a poco a poco codice basato su expl3, per esempio i nuovi "hooks" (si veda lthooks per maggiori informazioni).
Un esempio? Ecco: la definizione standard di `\par` è

```
\cs_new_protected:Npn \para_end: {
  \mode_if_horizontal:TF {
    \mode_if_inner:F {
         \tex_unskip:D
         \hook_use:n{para/end}
         \@kernel@after@para@end
         \mode_if_horizontal:TF {
           \if_int_compare:w 0 < \tex_lastnodetype:D
             \tex_kern:D \c_zero_dim
           \fi:
           \tex_par:D
           \hook_use:n{para/after}
           \@kernel@after@para@after
         }
         { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
    }
  }
  \tex_par:D
}
```

che non intendo spiegare, ma che mostra l'uso di expl3 nel kernel. Ci sono altre varianti, ma il kernel dice a un certo punto

```
\cs_set_eq:NN \par \para_end:
```

Anche la parte fondamentale di xparse è stata inserita nel kernel, dando accesso senza bisogno di altro alle funzionalità più potenti per definire comandi a livello utente.

G.P.: L'esempio che ci hai mostrato evidenzia chiaramente quello che, a tutta prima, sembra un deterrente all'uso di expl3: la verbosità autoesplicativa dei comandi. Non dico di essere affezionato alla stringatezza dell'assembly, ma al mio sguardo incompetente sembra di vedere un dizionario inverso: molte parole per indicare un unico comando. Ci spieghi da dove nasce l'esigenza di avere comandi cosí prolissi, qual è l'utilità e, se ci sono, quali gli svantaggi?

E.G.: Facciamo l'esempio di `\stepcounter`:

```
\def\stepcounter#1{%
  \addtocounter{#1}\@ne
  \begingroup
    \let\@elt\@stpelt
    \csname cl@#1\endcsname
  \endgroup}
```

Si deve sapere che la lista dei contatori associati al contatore `foo` è la macro `\cl@foo` la cui espansione è del tipo

```
\@elt{gnat}\@elt{gnu}
```

e che si ha

```
\def\@stpelt#1{\global\csname c@#1\endcsname \m@ne\stepcounter{#1}}%
```

Vediamone un'ipotetica implementazione in expl3:

```
\NewDocumentCommand{\stepcounter}{m}
 {
  \user_counter_step:n { #1 }
 }
\cs_new_protected:Nn \kernel_counter_step:n
 {
  \user_counter_incr:n { #1 }
  \seq_map_inline:cn { g__kernel_counter_#1_linked_seq }
   {
    \user_counter_set:n { ##1 } { -1 }
    \user_counter_step:n { ##1 }
   }
 }
```

con opportune definizioni delle funzioni richiamate. Ancora: per aggiungere un contatore alla lista di quelli associati si adopera internamente `\@addtoreset`

```
\def\@addtoreset#1#2{\expandafter\@cons\csname cl@#2\endcsname {{#1}}}
```

Come sarebbe in expl3? Più o meno

```
\cs_new_protected:Nn \user_counter_link:nn
 {
  \int_if_exist:cTF { g__user_counter_#1_int }
   {
    \seq_gput_right:cn { g__kernel_counter_#1_linked_seq } { #2 }
```

```
    }
    {
     \msg_error:nnn { user } { counter-not-exist } { #2 }
    }
  }
```

La variabile di tipo seq che contiene i contatori associati sarebbe allocata quando si esegue `\newcounter`

```
\def\newcounter#1{%
  \expandafter\@ifdefinable \csname c@#1\endcsname
    {\@definecounter{#1}}%
  \@ifnextchar[{\@newctr{#1}}{}}
```

che diventerebbe

```
\NewDocumentCommand{\newcounter}{mo}
 {
  \user_counter_new:n { #1 }
  \IfValueT{#2}{ \user_counter_link:nn { #1 } { #2 } }
 }
\cs_new_protected:Nn \user_counter_new:n
 {
  \int_if_exist:cTF { g__user_counter_#1_int }
   {
    \msg_error:nnn { user } { counter-exist } { #1 }
   }
   {
    \int_new:c { g__user_counter_#1_int }
    \seq_new:c { g__kernel_counter_#1_linked_seq }
    \cs_new:cn { the#1 } { \arabic{#1} }
    % più qualcos'altro ...
   }
 }
```

Più lungo, ma si sa in ogni riga ciò che si sta eseguendo e su che cosa.

Già che ci siamo, trovo che, rispetto a

```
\@nocounterr{#2}
```

il corrispondente expl3

```
\msg_error:nnn { user } { counter-not-exist } { #2 }
```

sia molto più chiaro.

All'inizio la verbosità può sconcertare, ma alla lunga la chiarezza paga. E l'uso del prefisso, qui `user` per indicare qualcosa che ha a che fare con l'interfaccia utente (è solo ipotetico) permette di distinguere al volo di che tipo sia la funzione da eseguire e dove andarla a cercare.

Perché `\@stpelt`? Eh, bisognerebbe chiederlo a Leslie Lamport. Credo che la parte `elt` venga dal Lisp, mentre `stp` sta per "step". Nome che rimane impresso nella memoria, vero?

G.P.: Indubbiamente questa chiarezza paga. C'è stato qualche linguaggio che abbia ispirato questa sintassi o è una scelta autonoma degli sviluppatori di "LaTeX3" (lo chiamo cosí per brevita)?

E.G.: Credo sia un'invenzione di Frank Mittelbach.

G.P.: A questo punto è opportuno sentire Frank Mittelbach. Torno tra pochissimo da te. Ciao Frank. È trascorso un po' di tempo dalla tua intervista in cui c'era anche Dave Walden. Felice di risentirti. Dunque, possiamo ritenere te l'artefice della sintassi dei comandi (a questo punto forse è meglio chiamarle istruzioni) di expl3?

FRANK MITTELBACH: Ti rispondo sinteticamente perché al momento sto lavorando all'uscita della terza edizione di *The LaTeX Companion*. E poi c'è il progetto sul *tagged* PDF che preme…Comunque, sì, sono largamente io, insieme ai componenti del team dell'epoca. Era circa il 1992. Impossibile ricordare chi ha portato come contributo cosa.

G.P.: Hai preso ispirazione da un linguaggio preesistente o è stata una scelta dettata da altre motivazioni? E perché?

F.M.: Ma, sai, si prende sempre ispirazione da ciò con cui hai lavorato, ma in questo caso direi certamente di no: l'ecosistema di TeX è così differente che il tradizionale concetto di linguaggio non è molto adatto. Se penso ai criteri di progettazione, li trovo molto "anti":

1. tieni lontano (e nascondi) il più possibile alcune delle stranezze di un linguaggio di espansione di macro in generale, e in particolare alcune delle stranezze di TeX;
2. sii molto diverso dall'usuale TeX;
3. sii molto strutturato (non sempre riuscito) ma permetti all'utente di intuire come invocare un comando senza fargli avere troppe sorprese.

Riguardo al punto 1., è stato fatto in modo che

- i trucchi di espansione fossero ridotti al minimo e relegati nelle porzioni di codice più interne;
- per la maggior parte del tempo fosse ragionevole pensare ai comandi come a funzioni e procedure e non a macro, con input e output chiari ed effetti diretti e collaterali ben definiti;
- l'espansione fosse per lo più gestita alterando la segnatura dell'argomento; ciò ha un meccanismo sempre uguale che può essere applicato a ogni funzione di base;
- i concetti fossero unificati quando possibile, così, per esempio, ... i comandi condizionali sono sempre forniti con argomenti TF (ma i predicati di basso livello esistono ancora quando servono per ragioni di velocità).

Riguardo al punto 2.

- abbiamo sempre visto esserci un miscuglio di codice (e sarà ancora più preminente con lo strato di programmazione L3 del formato LaTeX) e dovrebbe essere semplice distinguere cos'è expl3 e cosa no;
- aiuta a rimanere nello spirito: non usi dei truccacci di programmazione TeX se solo ti mantieni al codice di livello interface3;
- quando gli utenti si trovavano davanti a parecchi _ e : per ogni dove

Riguardo al punto 3.

G.P.: Grazie, Frank, per la tua disponibilità, nonostante il pochissimo tempo a tua disposizione, e buon lavoro per il tuo libro. Torniamo a Enrico. Hai chiarito perfettamente il salto di qualità espressiva del nuovo linguaggio ma, a quanto pare, l'utente finale non ne beneficerà direttamente. Quali prevedi saranno i benefici indiretti per questa classe di utenti?

E.G.: Non ci sono benefici "visibili" per l'utente finale, quello che si limita a scrivere documenti senza bisogno di definire più che qualche semplice comando. Se non consideriamo la maggiore robustezza delle implementazioni.

Per l'utente più smaliziato sono a disposizione molti nuovi metodi, fra cui gli "hook". Già ora è possibile adoperarli (occorre LaTeX del 2020-10-01 o successivo), un meccanismo che generalizza di parecchio i comandi di etoolbox per eseguire codice a ogni chiamata di un ambiente, ma non solo. Pacchetti come 'atbeginshi' sono diventati obsoleti, perché il nucleo include le loro funzionalità: l'ovvio vantaggio, a parte non dipendere da un pacchetto esterno, è che questi "hook" sono definiti in modo uniforme.

È assai probabile che altri pacchetti di uso frequente, penso a enumitem, per esempio, saranno poco alla volta integrati nel nucleo.

Il grosso problema è mantenere la compatibilità con i vecchi documenti, in cui ambienti e comandi sono spesso modificati agendo a basso livello. Ma quando il team decide qualche modifica importante, gli autori dei pacchetti che potrebbero essere resi instabili vengono contattati suggerendo che cosa cambiare. Se si seguono gli aggiornamenti su CTAN, si può vedere come questo accada abbastanza di frequente.

Piccoli passi. Sarebbe bello poter ridefinire `\parbox` con le nuove funzioni per snellire e semplificare l'implementazione, ma troppi documenti e codice esistente si appoggiano a comandi di livello più basso che sparirebbero.

In ogni caso, quando si decide per una modifica, il codice 'antico' non sparisce e si può adoperare latexrelease per tornare indietro. Un vecchio documento potrebbe non funzionare, ma basta aggiungere una chiamata a latexrelease per far tornare il nucleo a una versione precedente.

G.P.: Torniamo alle cassandre da cui è partita l'intervista, le quali prevedono un futuro breve per TeX e derivati e ne evidenziano una inutilità didattica. Credo che il tuo impegno su più fronti di sviluppo del "nostro" programma di composizione la dica lunga sul tuo pensiero in merito, ma vorrei comunque il tuo "disegno" sul futuro (o sui futuri) di questo programma che reputo complicato e affascinante. E quanto lungo prevedi questo futuro?

E.G.: Quale futuro ha TeX? Nell'ambito scientifico non dovrebbe aver problemi a sopravvivere. Molte case editrici (Springer in testa) forniscono classi per i loro libri e riviste; arXiv chiede che gli articoli siano sottoposti come TeX: sono centinaia ogni mese. Purtroppo, molti ambiti scientifici sono irraggiungibili ma non perché LaTeX non sarebbe un valido strumento: il problema è la corsa alla pubblicazione nel minor tempo possibile di montagne di articoli che nessuno leggerà mai, anche perché contengono poco più che risultati di un esperimento o di uno studio di portata limitata: qualche tabella generata da un foglio di lavoro, poco testo di commento sempre essenzialmente uguale, una bibliografia sterminata generata automaticamente. Il che non è un giudizio di merito, sia chiaro, solo un'amara constatazione.

Ma c'è un campo nel quale ci possono essere sviluppi molto interessanti: l'accessibilità. Da qualche mese la PDF Association, che si occupa di tutto quanto riguarda il formato PDF ha un gruppo di lavoro che riguarda proprio LaTeX ed è mirato a fornire gli strumenti per l'accessibilità. Il LaTeX team ne fa ovviamente parte. Lo scopo, forse ambizioso, è di rendere un documento PDF prodotto con LaTeX accessibile (nel senso tecnico, pensiamo a nonvedenti o altre categorie di persone che non possono maneggiare lo strumento nel modo usuale). Lo strato per l'accessibilità sarà creato nel modo più trasparente possibile, quindi con minimo intervento dell'autore del testo.

Se combiniamo con le capacità di programmazione proprie di LaTeX, molto più agevoli avendo a disposizione expl3, il progetto può dare nuova linfa al nostro sistema. E non è campato per aria: i primi risultati sono già disponibili anche se l'usabilità è ancora primitiva. Le ricadute saranno anche per gli utenti "normali": per esempio, hyperref sarà integrato nel formato, senza la necessità di modificare centinaia di comandi quando lo si vuole adoperare.

Il team di LuaTeX si muove in altre direzioni, non sempre così comprensibili. Tuttavia non vedo del tutto impossibile che il nuovo LuaMetaTeX diventi il motore "definitivo" (anche se c'è parecchio lavoro da fare).

In ogni caso, già oggi è possibile fare con XeLaTeX e LuaLaTeX cose che vent'anni fa sembravano al di là di Marte. E da quello che si vede nei siti che ne trattano, la base di utenti si espande. Le cassandre possono strillare quanto vogliono.

Ringrazio Enrico (e Frank) per aver esposto con voce autorevole una decisa precisazione alle affermazioni supportate da sensazioni e opinioni personali, ma non da dati di fatto, che tanto mi avevano colpito negativamente. Spero con questo di aver reso un servizio a quanti, come me, hanno avuto la stessa reazione.

Gianluca Pignalberi
(intervistatore)
g.pignalberi@gmail.com

Enrico Gregorio
(intervistato)
Enrico.Gregorio@univr.it

Frank Mittelbach
(intervistato)
frank.mittelbach@latex-project.org