Introduzione a arara

Enrico Gregorio*

18 febbraio 2013

Indice

1 Introduzione

Nei primi giorni di febbraio 2013 è stata pubblicata la versione 3 di arara con l'inclusione in TeX Live.

Il nome 'arara' è, in portoghese brasiliano, quello comune a ventitré specie dell'ordine *Psittaciformes*, cioè i pappagalli noti in italiano come 'ara'. Naturalmente l'argomento in discussione, cioè arara, non ha nulla a che fare con i pappagalli, ma con TeX e LaTeX. Il nome è stato scelto dall'autore, Paulo Cereda, che è brasiliano; per la versione 3 si sono aggiunti alla squadra di autori Marco Daniel (noto per **mdframed**) e Brent Longborough. Il progetto è *open source* e lo si può trovare su GitHub.[1]

Uno dei problemi più comuni agli utenti di LaTeX è quello di doversi ricordare tutte le manovre necessarie per avere un documento in forma finale:

```
pdflatex tesi
biber tesi
pdflatex tesi
pdflatex tesi
pdflatex tesi
```


[1] https://github.com/cereda/arara

supponendo che `tesi.tex` sia il nome del documento da elaborare. Se poi il documento richiede anche uno o più glossari, occorre aggiungere la chiamata di

```
makeglossaries tesi
```

e la lista potrebbe non essere finita qui. Per esempio, può essere necessario produrre il frontespizio con l'omonimo pacchetto, che richiederebbe i comandi

```
pdflatex tesi
pdflatex tesi-frn
pdflatex tesi
```

Esiste un potente programma (scritto in Perl) che è in grado di eseguire questi compiti da sé: **latexmk**. Tuttavia questo programma è una specie di scatola nera. Può essere configurato per tener conto di aspetti che non sono compresi nella modalità usuale, ma non sempre funziona.

La filosofia di **arara** è diversa. Non è lo sviluppatore del programma che decide quali compiti siano da svolgere, ma l'utente stesso. Uno dei limiti di **latexmk** è che questo programma lavora sempre supponendo che si debba costruire la versione finale, quindi con la risoluzione di tutti i riferimenti incrociati, la compilazione completa di indici, glossari, bibliografia e così via. Nel caso di **arara** si può invece decidere facilmente quali compiti eseguire: durante la preparazione del documento non è così impellente la necessità che i riferimenti bibliografici puntino esattamente al corretto documento nella bibliografia finale, né che i glossari siano perfetti. Normalmente si lancia la compilazione di `pdflatex` per controllare gli errori di sintassi LaTeX e poi leggere il documento in modo da verificare errori di ortografia o grammatica.

Nel nostro caso la soluzione con **arara** è di elencare i vari passi da eseguire tramite una lista di *direttive* da scrivere in testa al documento `.tex`:

```
% arara: pdflatex
% arara: frontespizio
% arara: makeglossaries
% arara: biber
% arara: pdflatex
% arara: pdflatex
% arara: pdflatex
%
\documentclass[a4paper]{book}
```

Nel caso non si reputi necessario eseguire tutte quelle direttive, basterà commentarle, per esempio con

```
% arara: pdflatex
%! arara: frontespizio
%! arara: makeglossaries
%! arara: biber
%! arara: pdflatex
%! arara: pdflatex
%! arara: pdflatex
%
\documentclass[a4paper]{book}
```

arara eseguirà solo la prima.

2 Come cominciare

Assumeremo che arara sia disponibile nella distribuzione TEX, cioè che si abbia una TEX Live 2012 (o successiva) aggiornata al 10 febbraio 2013. C'è anche il modo di installare arara in modo indipendente, ma non vale la pena di fare fatica. È necessario avere sulla propria macchina Java in versione almeno 1.5; difficile che non ci sia. Questo è un requisito obbligatorio e non si può fare nulla se Java manca.

Non ci si spaventi di fronte alla temibile *linea di comando*; la usiamo qui come esempio e daremo poi vari modi per poterne fare a meno, integrando i comandi in vari *front end* a LATEX.

Scriviamo un semplice documento di esempio, un classico da scuola elementare,

```
% arara: pdflatex
\documentclass[a4paper]{article}
\begin{document}
Hello world
\end{document}
```

e chiamiamolo `test.tex`. Si noti la prima riga, che sebbene sia commentata è decisiva. Dalla linea di comando eseguiamo

```
arara test
```

e il risultato sul terminale sarà

```
  __ _ _ __ __ _ _ __ __ _
 / _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
 \__,_|_|  \__,_|_|  \__,_|
Running PDFLaTeX... SUCCESS
```

Se il sistema operativo è predisposto per l'italiano, si vedrà

```
  __ _ _ __ __ _ _ __ __ _
 / _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
 \__,_|_|  \__,_|_|  \__,_|
Sto eseguendo PDFLaTeX... SUCCESSO
```

Che significa? Che il nostro documento è stato composto senza errori! Se però aggiungiamo nel documento l'inesistente comando `\ddt`, otterremo un risultato diverso:

```
  __ _ _ __ __ _ _ __ __ _
 / _` | '__/ _` | '__/ _` |
| (_| | | | (_| | | | (_| |
 \__,_|_|  \__,_|_|  \__,_|
Running PDFLaTeX... FAILURE
```

Sembra un po' scarno e, in effetti, lo è. Ma possiamo ottenere il solito comportamento con la chiamata

```
arara --verbose test
```

e l'errore inserito farà interrompere il processo con il solito messaggio

```
! Undefined control sequence.
l.4 Hello world\ddt
```

e la possibilità di inserire qualche correzione o di premere 'invio' per proseguire.

Dov'è il vantaggio rispetto a eseguire direttamente

```
pdflatex test
```

ci si potrebbe chiedere. Ci sono *parecchi* vantaggi. Per esempio, potremmo voler abilitare la sincronizzazione fra documento LaTeX e il documento PDF finale. Dovremmo ricordarci ogni volta di lanciare

```
pdflatex -synctex=1 test
```

mentre con arara basta modificare la direttiva una volta per tutte in

```
% arara: pdflatex: { synctex: yes }
```

perché ogni volta che lanciamo `arara test` l'opzione di sincronizzazione sia espressa. Un'altro vantaggio, forse non così decisivo, ma che può rivelarsi utile. Aggiungiamo una nuova direttiva così:

```
% arara: pdflatex: { draft: yes }
% arara: pdflatex: { synctex: yes }
\documentclass{article}
...
```

Ogni esecuzione di `arara test` eseguirà due volte `pdflatex` sul nostro documento, la prima volta con l'opzione `-draftmode`, la seconda con `-synctex=1`. L'opzione `-draftmode` non produce alcun PDF ed è utile per controllare eventuali errori di sintassi: se ce ne sono, arara terminerà l'esecuzione alla prima direttiva, senza eseguire la seconda.

Questo è un fatto generale: un errore nell'esecuzione di una direttiva interrompe il programma, cosa che non capita con **latexmk**. Se lanciamo arara con l'opzione `--verbose`, potremo correggere al volo gli errori, ricordandoci poi di controllare il log e di correggere il documento LaTeX, senza dover aspettare che altre direttive ci facciano perdere tempo.

Quando sapremo che il codice del nostro documento è privo di errori sintattici, potremo anche fare a meno della prima direttiva; semplicissimo come aggiungere un solo carattere:

```
%! arara: pdflatex: { draft: yes }
% arara: pdflatex: { synctex: yes }
\documentclass{article}
...
```

Una riga che non sia della forma

```
% arara: ⟨regola⟩
```

sarà del tutto ignorata e quindi il semplice `!` fa al caso nostro. Possiamo tornare indietro facilmente e abbiamo tutto sotto controllo nel nostro documento LaTeX, non in file di configurazione come per **latexmk** o un `Makefile` per **make**.

Un'altra opzione per arara è `--log` che registra in un file di nome `arara.log` tutta la sessione e si rivela *molto* utile nel caso in cui Biber decida di non voler più funzionare, cosa che accade in modo abbastanza casuale, ma fastidioso. Nel file `arara.log` si troverà qualcosa del tipo

```
11 Feb 2013 14:13:54.562 INFO  CommandTrigger - 'PDFLaTeX' was successfully executed.
11 Feb 2013 14:13:54.562 INFO  CommandTrigger - Running 'Biber'.
```

```
11 Feb 2013 14:13:54.562 TRACE CommandTrigger - Command: biber  "tesi"
11 Feb 2013 14:13:56.121 TRACE CommandTrigger - Output logging:
11 Feb 2013 14:13:56.121 TRACE CommandTrigger - data source
 /var/folders/gF/gFRYtF2o2RWQLU+8ZM8gX++++Tc/-Tmp-/par-656e7269636f6d62/
 cache-eeddb72d39f441eb365adf9a751662cebe94deff//inc/lib/Biber/LaTeX/
 recode_data.xml not found in .
```

e sapremo che dovremo trovare la cartella

```
/var/folders/gF/gFRYtF2o2RWQLU+8ZM8gX++++Tc/-Tmp-/par-656e7269636f6d62
```

e toglierla di mezzo senza preoccupazioni. Sui diversi sistemi operativi il percorso sarà differente, ma in ogni caso dovrebbe essere chiaro qual è la cartella da cancellare, perché la parte `//inc/lib/Biber/...` ci dovrebbe essere sempre.

3 Regole

La distribuzione standard di **arara** contiene un buon numero di regole predefinite. Quasi tutte hanno la possibilità di aggiungere opzioni; se la regola si chiama `foo`, avremo le due possibilità

```
% arara: foo
% arara: foo: { ⟨opzione⟩ , ⟨opzione⟩ , ... }
```

Si noti che nel secondo caso occorrono i due punti, che segnalano ad **arara** di continuare a leggere; nel primo caso *non* si devono mettere i due punti.

Non entreremo nel dettaglio della sintassi per la quale rimandiamo al manuale. Ci interessa illustrare le più utili forme delle regole e vediamo prima l'elenco delle principali:

- `pdflatex`
- `xelatex`
- `lualatex`
- `biber`
- `bibtex`
- `frontespizio`
- `makeglossaries`
- `nomencl`
- `makeindex`

Non è difficile capire che fanno le prime tre: lanciano la compilazione con le tre varianti di LaTeX. La quarta e la quinta si occupano di lanciare Biber o BibTeX per la creazione della bibliografia (Biber è quello da usare con **biblatex**).

Chi adopera il pacchetto **frontespizio** sarà lieto di sapere che la regola si occupa di compilare il file ausiliario in modo che la successiva compilazione con LaTeX trovi il PDF del frontespizio per l'inclusione: niente più problemi con il file `-frn` da trovare, aprire e compilare, ci pensa **arara**!

Analogamente, chi usa **nomencl** sa che per predisporre l'elenco della nomenclatura è necessario un comando piuttosto oscuro:

```
makeindex tesi.nlo -s nomencl.ist -o tesi.nls
```

sempre ammesso che `tesi.tex` sia il nome del nostro documento. Niente più ricerche affannose del giusto incantesimo: basta dare la direttiva

```
% arara: nomencl
```

e **arara** si occupa di tutto. Lo stesso per `makeglossaries` che va fatto girare quando si adopera il pacchetto **glossaries**. La chiamata è semplice,

```
makeglossaries tesi
```

ma è noioso doversene ricordare e, per alcuni, dover agire con la linea di comando. Naturalmente tutti sanno che il pacchetto **imakeidx** evita la necessità di chiamare il programma esterno MakeIndex, ma può essere comodo in alcune situazioni avere a disposizione anche questa regola.

Ci sono anche le regole `latex`, `dvips` e `ps2pdf` per quei documenti che usano **PSTricks**: il solito complicato ciclo

```
latex tesi
dvips tesi.dvi -o
ps2pdf tesi.ps
```

inframmezzato dall'esecuzione di altri programmi come BibTeX o Biber, diventa il più semplice

```
% arara: latex
% ⟨altre direttive⟩
% arara: dvips
% arara: ps2pdf
\documentclass[a4paper]{book}
```

4 Opzioni

Vediamo un tipico insieme di direttive per un documento che richieda solo di compilare la bibliografia con BibTeX:

```
% arara: pdflatex
% arara: bibtex
% arara: pdflatex
% arara: pdflatex: { synctex: yes }
```

Nell'ultima direttiva abbiamo aggiunto un'opzione alla regola `pdflatex`. Non c'è bisogno di attivare la sincronizzazione per tutte le compilazioni, basta l'ultima; così la attiviamo solo per il gran finale.

Alcuni pacchetti, come **minted** o **abc**, hanno bisogno che sia abilitata la cosiddetta *shell escape*, cioè la possibilità di eseguire programmi esterni durante la compilazione. L'opzione da passare a `pdflatex` è `shell: yes`. In tal caso le direttive di prima diventerebbero

```
% arara: pdflatex: { shell: yes }
% arara: bibtex
% arara: pdflatex: { shell: yes }
% arara: pdflatex: { shell: yes , synctex: yes }
```

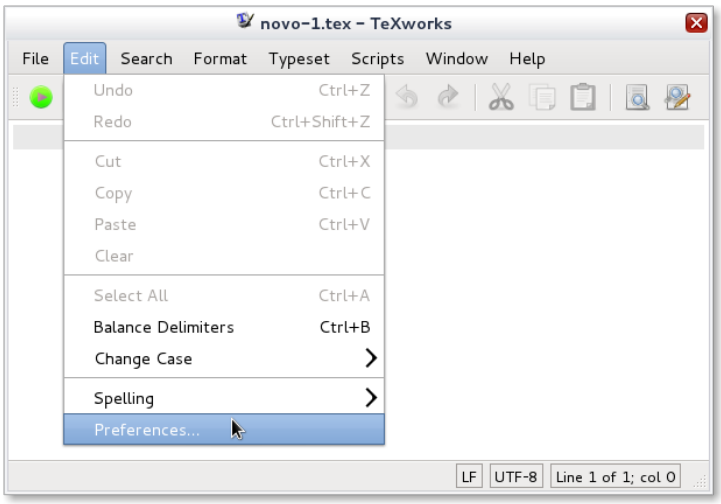

Figura 1. Preferenze di TEXworks

Possiamo dunque passare qualsiasi numero di opzioni (purché sensate) in ciascuna direttiva.[2]

Si potrebbe avere la tentazione di abilitare la *shell escape* per ogni documento, ma è meglio evitarlo e usarla solo per documenti e pacchetti affidabili.

5 Come eseguire arara?

Non tutti, anzi pochi, adoperano la linea di comando per lanciare la compilazione dei propri documenti. Vediamo come attivare in alcuni editor di uso comune la possibilità di compilare tramite arara.

5.1 TEXworks

La procedura è molto semplice. Si lanci TEXworks e si vada sulle preferenze del programma (figura 1) e si scelga il tab "Typesetting" (figura 2).

Si prema il pulsante "+" accanto al box "Processing tools"; la finestra che si apre va completata secondo quanto si vede nella figura 3, eccetto che `/usr/local/arara/arara` va sostituito con il solo `arara`. Per aggiungere argomenti (uno per riga), si prema il pulsante "+". Si consiglia l'opzione `--verbose`, come nella figura; l'opzione `--log` registra in un file `arara.log` tutta la sessione e vale la pena di usarla.

Premendo il pulsante OK in questa finestra e in quella delle preferenze, si tornerà alla finestra principale e nel menù a discesa accanto alla freccia per la compilazione potremo scegliere arara, come nella figura 4.

Se precediamo la lista delle direttive per arara con la riga magica, il motore sarà scelto automaticamente all'apertura del file:

```
% !TEX program = arara
% arara: pdflatex
% <altre direttive>
```

[2] Attenzione: ogni direttiva deve essere su una sola riga, per quanto lunga; è possibile che nuove versioni di arara superino questa limitazione.

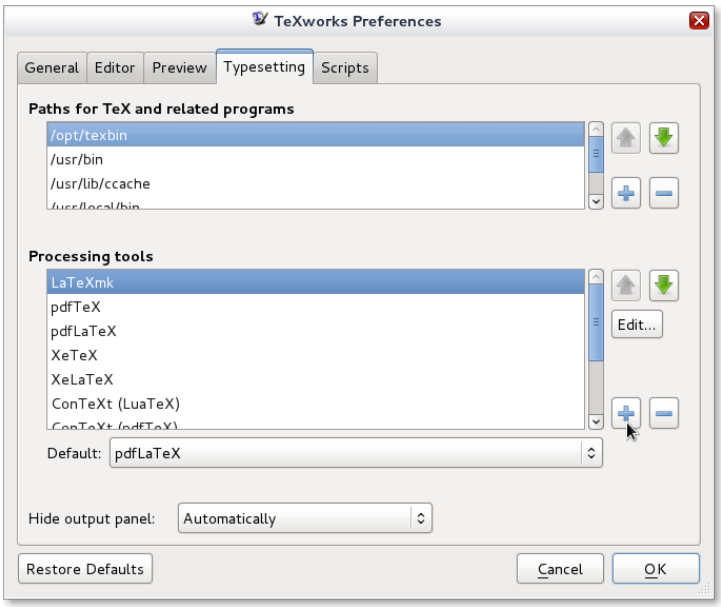

Figura 2. Preferenze di TEXworks, tab "Typesetting"

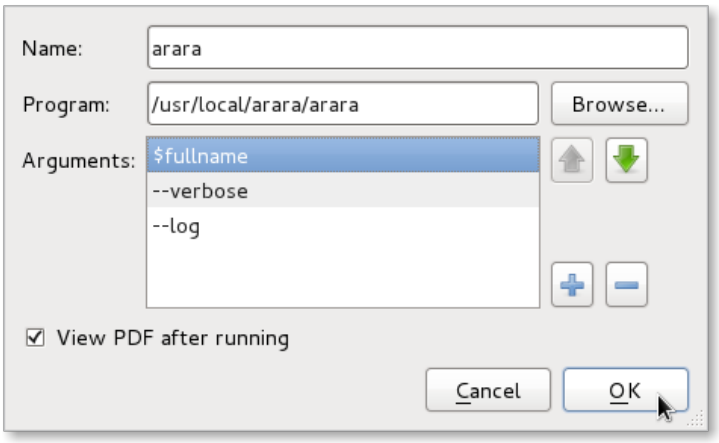

Figura 3. Definire il metodo di compilazione

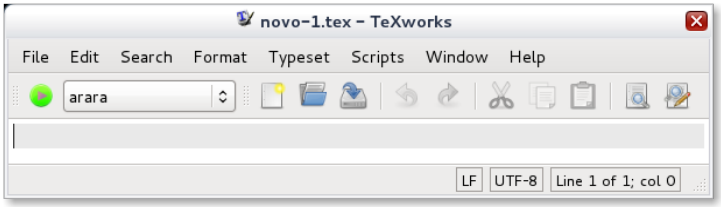

Figura 4. Il menù a discesa riporta arara

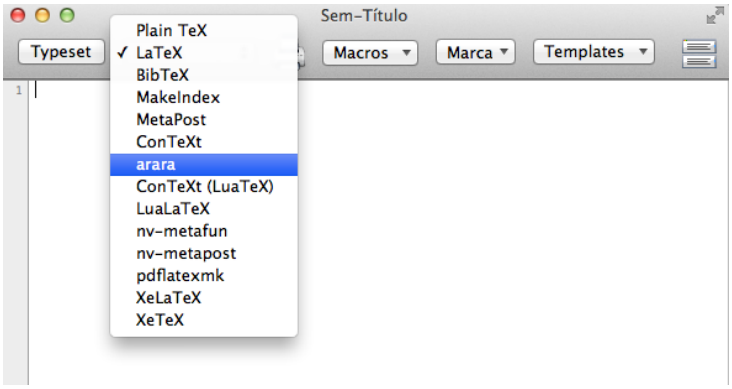

Figura 5. TEXShop e arara

```
%
\documentclass[a4paper]{article}
...
```

5.2 TEXShop

Si spera che un prossimo aggiornamento di TEXShop conterrà già il metodo per arara. Nel frattempo è semplice costruirsene uno. Si vada nella cartella

⌂/Library/TeXShop/Engines

e si faccia una copia di `XeLaTeX.engine` chiamandola `arara.engine`. Si lanci TEXShop e si apra il file `arara.engine`, sostituendo il suo contenuto con

```
#!/bin/bash
export PATH=${PATH}:/usr/texbin:/usr/local/bin
arara --verbose --log "$1"
```

Si chiuda TEXShop e lo si rilanci. Ora nel menù a discesa per scegliere il motore di composizione dovremmo trovare arara, come nella figura 5. L'opzione `--log` è facoltativa, ma consiglio di usarle entrambe.

Anche per TEXShop è possibile attivare automaticamente la scelta di arara con la riga magica

```
% !TEX TS-program = arara
% arara: pdflatex
% <altre direttive>
%
\documentclass[a4paper]{article}
...
```

A differenza di quanto accade con TEXworks, il motore scelto con la riga magica non compare automaticamente nel menù a discesa, ma funziona lo stesso: quello che compare nel menù è prevalente solo quando non è il primo della lista.

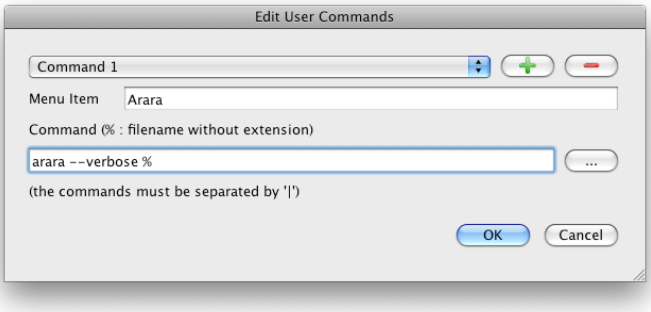

Figura 6. Definizione di un comando per TEXStudio

5.3 TEXStudio, Texmaker, Kile

In TEXStudio, Texmaker e Kile è possibile senza troppi sforzi costuirsi nuovi meccanismi per compilare tramite arara. Per esempio in TEXStudiobasta scegliere il menù

User > User Commands > Edit User Commands

e impostare un comando come indicato nella figura 6, premendo prima il pulsante "+". Il programma assegnerà anche una scorciatoia da tastiera. Oltre all'opzione `--verbose` è consigliabile usare anche `--log`.