# I comandi condizionali

*Claudio Beccari*

## Sommario

I comandi condizionali sono raramente usati dagli utenti LaTeXe quelle poche volte che lo fanno ricorrono, per evitare alcune loro trappole, a pacchetti esterni. Tali pacchetti però non sempre sono utili allo scopo. Fronteggiare gli errori che si presentano richiede una conoscenza approfondita di ciò che sta sotto ai comandi condizionali, al fine di servirsi di quelli più adatti e più in generale dei pacchetti che li definiscono.

## Abstract

Conditional commands are seldom used by LaTeX users and in those rare instances they resort to external packages that are supposed to skip the insidious traps set up by these commands. Some such packages sometimes don't succeed in this task. In order to master the unusual error situations that may arise, it is better to know what there is under the hood of the typesetting engine conditional command interpretation so as to use them or the right packages in the proper way.

## 1 Introduzione

I comandi condizionali sono quelli che cominciano con `\if`; ce ne sono molti, predefiniti già dagli albori di TeX e accessibili direttamente dagli utenti di LaTeX; questo a sua volta ne definisce altri mediante una sintassi più robusta. Inoltre le estensioni di TeX, grazie a ε-TeX, sono già incorporate nei motori di composizione *pdftex* dal lontano 2005 ed analogamente nei motori di composizione *xetex* e *luatex* fin dalla loro nascita. Questi nuovi comandi estendono la possibilità di verificare diversi altri eventi, cosa difficilmente controllare con i comandi originali.

I comandi condizionali normalmente richiedono la sintassi seguente, nonostante altri comandi possano venire definiti in modo diverso:

```
\if⟨condizionale⟩
    ⟨azione se vero⟩
\else
    ⟨azione se falso⟩
\fi
```

A loro volta le azioni ⟨*azione se vero*⟩ e ⟨*azione se falso*⟩ possono contenere altre espressioni condizionali complete. Questo fatto però, come vedremo più avanti, rende questi costrutti relativamente fragili, in particolare quando essi sono complicati oppure quando particolari azioni sono da compiere.

### 1.1 I comandi di TeX

I comandi oggi disponibili a livello di interprete (cioè definiti nel motore di composizione, non nel nucleo di LaTeX) sono i seguenti (Knuth, 1996; Breitenlohner, 1998):

`\if` confronta due token; questi possono essere dei singoli caratteri espliciti, delle macro, argomenti di macro; prima di eseguire i confronti questi token vengono eventualmente sviluppati; se lo sviluppo di una macro contiene a sua volta più token, il confronto viene fatto fra i primi due.

`\ifcat` confronta i codici di categoria di due token che rappresentano caratteri espliciti, argomenti di macro, macro, e simili; le macro vengono sviluppate prima che venga eseguito il confronto. Si noti la differenza fra `\if` e `\ifcat`; il secondo confronta i codici di categoria di due caratteri, il primo confronta i codici interni di due caratteri, quasi sempre coincidenti con i codici ascii.

`\ifnum` esegue un confronto numerico fra due entità rappresentate da contatori, numeri espliciti, macro, caratteri impliciti, argomenti di macro, eccetera; le due quantità da confrontare sono separate da un operatore di relazione fra i seguenti: `=`, `>`, `<`. Prima di eseguire il confronto le macro vengono sviluppate e i contenuti dei contatori vengono recuperati, in modo che i confronti avvengano realmente fra numeri.

`\ifdim` simile a `\ifnum`, ma confronta due lunghezze; di nuovo gli elementi da confrontare possono venire indicati con macro, registri lunghezza, lunghezze esplicite, eccetera. I due token sono separati da due operatori di relazione come per i controlli numerici.

`\ifodd` controlla se lo sviluppo di quello che segue rappresenta un valore numerico intero dispari.

`\ifvmode` controlla se il modo di comporre corrente sia quello verticale.

`\ifhmode` controlla se il modo di comporre corrente sia quello orizzontale; in LaTeX questo modo è anche chiamato *LR mode*.

`\ifmmode` controlla se si sta componendo in modo matematico.

`\ifinner` controlla se si sta componendo in modo 'interno'; questo capita per esempio quando si sta componendo testo all'interno di una scatola; per gli utenti di LaTeX vale la pena di ricordare che quando si compone all'interno degli ambienti figure e table si è in modo interno.

`\ifvoid` controlla se un token che rappresenta il nome di una scatola o il suo numero identificativo esplicito indica una scatola completamente vuota.

`\ifhbox` controlla se una scatola contenga materiale orizzontale

`\ifvbox` controlla se una scatola contenga materiale verticale.

`\ifx` controlla se due macro hanno le stesse caratteristiche per quanto riguarda la loro natura globale e se siano entrambe 'long'[1] e se la loro definizione è coincidente; le due macro *non* vengono sviluppate.

`\ifeof` controlla se nel leggere un file si è raggiunta la sua fine (*end of file*); quando si cerca di leggere un file inesistente, la sua inesistenza equivale ad un *end of file*; viene usato per esempio in comandi LaTeX del tipo `\IfFileExists` o `\InputIfFileExists` ma può venire usato da chiunque.

`\iftrue` è un test particolare perché restituisce sempre il valore 'vero'.

`\iffalse` analogamente questo è un test particolare che restituisce sempre il valore 'falso'.

`\ifcase` è un test soggetto a un contatore; verifica il contenuto del contatore, che supponiamo contenga il valore $n$, poi esegue fra un elenco di azioni possibili, l'$n$-esima azione; le azioni della lista sono separate mediante il comando `\or`; se non si vuole esaminare la lista oltre un certo punto si può usare il comando `\else` e definire un'azione di default; in ogni caso la lista viene terminata mediante il comando `\fi`.

`\ifdefined` controlla se un token è definito; tipicamente il token è una macro o una 'control sequence' a cui può essere stato assegnato un significato con `\let`.

`\ifcsname` controlla se il nome di una 'control sequence' corrisponde ad una macro definita; il nome della macro viene esplicitato senza il backslash davanti e viene terminato mediante `\endcsname`. Questo controllo è più efficace di quello con la vecchia sintassi; si noti infatti che i seguenti due costrutti (vecchio e nuovo) sono equivalenti:

```
\expandafter
  \csname␣⟨nome⟩\endcsname\relax
    ⟨azione se falso⟩
\else
    ⟨azione se vero⟩
\fi

\ifcsname␣⟨nome⟩\endcsname
    ⟨azione se vero⟩
\else
    ⟨azione se falso⟩
\fi
```

`\iffontchar` viene seguito da un nome di font, oppure dal comando `\font` che rappresenta il nome del font corrente, e poi da un numero non superiore a 255, per verificare se il glifo con quell'indirizzo dentro quel font esiste davvero. L'indirizzo 255 è il massimo che utilizzabile con *pdftex* il quale accetta solo font di tipo Type 1 o comunque senza caratteri oltre quell'indirizzo. Con i motori *xetex* e *luatex*, che gestiscono font UNICODE contenuti in polizze ricchissime di caratteri di ogni genere, il numero corrispondente all'indirizzo può essere virtualmente grande 'a piacere', senza superare la capacità di gestione dei numeri interi da parte del sistema TeX.

`\unless` questo non è un vero comando condizionale, ma quando precede un comando condizionale ne nega l'esito; quindi, se senza `\unless` un comando condizionale fornisce il risultato 'vero', allora preceduto da `\unless` fornisce il risultato 'falso'.

Altri comandi condizionali possono venire definiti mediante il comando `\newif` e si comportano praticamente come i comandi di TeX. Si tenga presente che il comando `\newif` definisce il test logico il cui nome comincia con `\if`, ma definisce anche due altri comandi per assegnare il valore vero o falso a questo comando; per esempio, se volessimo definire la variabile booleana `blu` per controllarne il valore mediante il test logico `\ifblu`, allora si scriverebbe:

```
\newif\ifblu
```

Appena viene definito, il test è 'falso' ma `\newif` ha definito anche i comandi `\blutrue` e `\blufalse` con cui assegnare il valore 'vero' o 'falso' al test logico `\ifblu`; di fatto i comandi in questione sono:

```
\def\blutrue{\let\ifblu\iftrue}
\def\blufalse{\let\ifblu\iffalse}
```

1. Una macro 'long' accetta come argomenti anche più capoversi; una macro che non abbia questa caratteristica non accetta che una frazione di capoverso.

così che di volta in volta essi rendono il comando `\ifblu` equivalente ai test nativi `\iftrue` o `\iffalse` visti sopra.

### 1.2 I comandi di LaTeX

Altri comandi condizionali definiti mediante il comando `\newif` vengono di solito definiti nei file di classe; per esempio la classe `book` definisce anche i comandi condizionali relativi alle opzioni che può gestire; i significati e gli scopi traspaiono abbastanza dai loro nomi;

```
\if@restonecol          \if@titlepage
\if@openright           \if@mainmatter
\if@compatibility       \if@twoside
\if@twocolumn
```

I comandi condizionali di LaTeX sono definiti mediante macro che a loro volta si appoggiano ai condizionali nativi del linguaggio TeX, ma sono costruiti in modo da essere più robusti. Tali comandi condizionali sono definiti all'interno del nucleo di LaTeXe sono 'protetti', nel senso che, contenendo il carattere '@' che normalmente non è una lettera, evitano che l'utente possa inavvertitamente ridefinire un comando interno. Essi sono indicati nell'elenco che segue senza descriverne la sintassi, sebbene essa traspaia abbastanza dal nome.

```
\@ifnextchar            \@ifstar
\@ifdefinable           \@ifundefined
\@ifforloop             \@iffileonpath
\@ifframebox            \@iframepicbox
\@ifatmargin            \@ifpackageloaded
\@ifclassloaded         \@ifpackagelater
\@ifclasslater          \@ifl@aded
\@ifl@ter               \@ifl@t@r
\@ifpackagewith         \@ifclasswith
\@if@ptions             \@if@pti@ns
```

In generale questi comandi interni di LaTeX richiedono due argomenti (fra graffe, se non sono costituiti da un solo token) che contengono rispettivamente l'⟨*azione se vero*⟩ e l'⟨*azione se falso*⟩. Talvolta questi due argomenti possono essere preceduti da un ulteriore primo argomento; infatti all'interno del nucleo di LaTeX i comandi sono definiti in modo da accorgersi se il primo argomento è facoltativo, ovvero racchiuso fra parentesi quadre, o se gli argomenti che seguono il comando sono solo quelli obbligatori; per esempio, è nota la sintassi del comando matematico per indicare la radice:

`\sqrt[`⟨*indice*⟩`]{`⟨*radicando*⟩`}`

L'⟨*indice*⟩ della radice è chiaramente facoltativo, perciò la definizione del comando `\sqrt` deve verificare se il primo token che lo segue (*next character*, un singolo token) sia o meno una parentesi quadra; infatti la definizione dentro il nucleo di LaTeX (contenuto nel file `latex.ltx`) è la seguente:

```
\DeclareRobustCommand\sqrt{%
    \@ifnextchar[\@sqrt\sqrtsign
}
```

Si vede dunque che il condizionale `\@ifnextchar` è seguito da tre token; il primo è il carattere da verificare, la parentesi quadra aperta, il secondo è il comando da eseguire se il risultato del confronto è vero, mentre il terzo indica l'azione da eseguire se il confronto è falso.

## 2 La fragilità dei comandi

Fin dai suoi albori i comandi dell'interprete TeX e di quelli definiti nel primo insieme di macro `plain.tex` sono risultati affetti dalla fragilità; oggi il LaTeX 3 Team ha apportato moltissime correzioni al nucleo di LaTeX in modo da eliminare quasi completamente la presenza di comandi fragili, tanto che oggi è difficile incontrarne; ma se l'utente scrive le sue macro ricorrendo ai comandi dell'interprete, allora è possibile che egli stesso crei macro fragili.

Una macro è robusta quando il suo testo sostitutivo, cioè la sua definizione, viene sempre sviluppato ed eseguito correttamente e completamente in qualsiasi circostanza; altrimenti il comando è fragile.

La guida di Leslie Lamport (Lamport, 1994) (come la sua traduzione commentata (Beccari, 2014a)) riportano un elenco di comandi LaTeX fragili; non sono moltissimi, ma sono di uso frequente; tra loro il comando `\\` che accetta ben due argomenti facoltativi, l'asterisco e l'ammontare della spaziatura verticale indicata fra parentesi quadre.

La fragilità è dovuta a diverse cause, la più importante delle quali è costituita dallo sviluppo incompleto delle varie macro invocate dal comando fragile; ciò può dipendere dal fatto che una parte dello sviluppo venga scritta nei file ausiliari, oppure che i vari `\if`, `\else` e `\fi` che costituiscono il costrutto "primitivo" di comandi condizionali, non vengono eseguiti al momento giusto. Alcune delle soluzioni che spesso risolvono i problemi di fragilità sono indicate nei paragrafi seguenti.

## 3 Cenni al linguaggio L3

Il linguaggio L3 è in fase di sviluppo da parte del LaTeX 3 Team per costituire la base per la definizione del nuovo LaTeX 3. Questo, dopo una ventina d'anni dalla decisione di realizzare la versione 3 del nuovo nucleo di LaTeX, comincia ad essere disponibile in via sperimentale.

L'idea fondante del linguaggio L3 è quella di costituire un'interfaccia fra i comandi definiti dall'utente e l'interprete sottostante in modo che si possano gestire correttamente e in modo robusto tutte le possibili strutture informatiche che un programma per la composizione tipografica può richiedere, con particolare attenzione alla gestione degli argomenti facoltativi e dei comandi condizionali.

Alcune parti di questo linguaggio sono già disponibili e vengono normalmente distribuite con le installazioni moderne. Diversi pacchetti di estensione del linguaggio LaTeX sono già scritti e molti tra loro, specifici per XeLaTeX e LuaLaTeX, sono già stati realizzati; in questo articolo, però, mi limiterò a descrivere i comandi ed i pacchetti che verosimilmente l'utente userà più frequentemente con pdfLaTeX.

## 4 I pacchetti utili

Di per sé i comandi del linguaggio originale TeX non sono difficili da usare, ma hanno indubbi inconvenienti: (*a*) la fragilità; (*b*) l'impossibilità di usare "frasi logiche"; (*c*) l'evidente complessità nel caso si debbano fare controlli molto complicati.

## 5 Le estensioni del pacchetto babel

Il pacchetto babel (Bezos, 2013) mette a disposizione due comandi `\bbl@afterelse` e `\bbl@afterfi` che permettono di eseguire l'⟨*azione se vero*⟩ o l'⟨*azione se falso*⟩ scavalcando i comandi `\else` o `\fi`. Questi comandi usano argomenti delimitati e in un certo senso sono più efficaci ed efficienti dei comandi nativi del linguaggio TeX, ma sono altrettanto e forse anche più fragili di questi ultimi; in particolare richiedono la creazione di gruppi per poter costruire dei test annidati.

Dentro babel funzionano bene, ma ne sconsiglierei fortemente l'uso per definire macro personali.

## 6 Il pacchetto ifthen

Il pacchetto ifthen (Carlisle, 2001) affronta questi problemi in modo più efficace; la sintassi è la seguente:

```
\ifthenelse{⟨test⟩}%
{⟨azione se vero⟩}     {⟨azione se falso⟩}
```

dove ⟨*test*⟩ è un test logico che produce come risultato uno stato di 'vero' o 'falso'. Il pacchetto prevede ⟨*test*⟩ per eseguire quasi tutti i controlli eseguibili con i comandi condizionali nativi di TeX; ma ⟨*test*⟩ può essere anche una frase logica, racchiusa fra i delimitatori `\(` e `\)`, che collega fra loro diversi test mediante i connettori logici `\NOT`, `\OR` e `\AND`. Chiunque conosca i rudimenti di logica booleana delle scuole secondarie può ragionevolmente scrivere una frase logica corretta senza avere una particolare esperienza di programmazione. Questo pacchetto è ulteriormente esteso da xifthen (Noirel, 2009) che accetta una sintassi più efficace e robusta per la costruzione di frasi logiche.

Prescindiamo per ora dai ⟨*test*⟩ specifici che si possono eseguire e cerchiamo di capire perché la sintassi del pacchetto ifthen è quasi sempre robusta. Sostanzialmente la lunga e complessa definizione del comando principale `\ifthenelse` assegna il valore 'vero' o 'falso' in base al risultato di ⟨*test*⟩ al comando condizionale `\ifTE@val`, che è reso equivalente a `\iftrue` oppure a `\iffalse`.

Alla fine della definizione di `\ifthenelse` compare il test finale:

```
\ifTE@val
   \expandafter\@firstoftwo
\else
   \expandafter\@secondoftwo
\fi
```

È proprio qui che si trova il congegno che rende robusto il comando `\ifthenelse`; esso non legge gli ultimi due argomenti della sua sintassi, si limita ad eseguire il ⟨*test*⟩. La scelta se eseguire la prima o la seconda azione viene affidata a quei due comandi 'misteriosi' che compaiono dopo `\expandafter`: `\@firstoftwo` e `\@secondoftwo`. LaTeX nel suo nucleo ricorre spesso a questo genere di comandi.

Le definizioni di questi due comandi corrispondono al loro nome, accettando due argomenti ma selezionando solo il primo oppure il secondo argomento:

```
\def\@firstoftwo#1#2{#1}
\def\@secondoftwo#1#2{#2}
```

È proprio questo meccanismo che permette a `\ifthenelse` di essere praticamente robusto. Ma la sintassi di `\ifthenelse` si è mostrata fragile almeno in un caso che mostrerò più avanti nonostante ciò nulla tolga all'affidabilità del pacchetto `\ifthen`.

Vediamo un caso che riprendo dalla cosiddetta Guida GuIt (Beccari, 2014b).

Il comando `\cleardoublepage` è definito nel nucleo di LaTeX[2] mediante l'uso dei comandi nativi di TeX:

```
\def\cleardoublepage{\clearpage
    \if@twoside\ifodd\c@page\else
    \hbox{}\newpage\if@twocolumn
    \hbox{}\newpage\fi\fi\fi}
```

Sono riconoscibili i comandi condizionali `\if@twocolumn` e `\if@twoside` usati anche nella classe book ma definiti nel nucleo di LaTeX; si vede all'opera anche il comando primitivo `\ifodd`.

Usando il pacchetto ifthen questa definizione potrebbe essere tradotta in:

```
\newcommand*\cleardoublepage%
{\clearpage
  \ifthenelse{%
  \(% inizio frase logica
    \boolean{@twoside}%
    \AND\NOT\isodd{\value{page}}%
  \)% fine frase logica
```

2. Qui se ne riporta una versione semplificata.

```
  }{%
     \mbox{}\newpage
     \ifthenelse{\boolean{@twocolumn}}%
     {\mbox{}\newpage}{}%
  }{}%
}
```

Come si vede i comandi condizionali definiti con `\newif`, come `\if@twoside`, nel nucleo di LaTeX, vengono controllati per conoscere il loro stato di 'vero' o 'falso' mediante il comando `\boolean` il cui argomento è la variabile booleana stessa, cioè il nome del comando condizionale privato del prefisso `\if`. I connettori logici `\AND` e `\NOT` servono per definire una frase logica che connette due diversi stati di 'vero' o 'falso'; il test `\isodd` usa una sintassi leggermente diversa, ma corrisponde al comando condizionale `\ifodd`.

Il pacchetto ifthen, nell'esempio mostrato, con un solo comando `\ifthenelse` permette l'esecuzione della medesima operazione che richiede con i comandi nativi di TeX tre costrutti condizionali annidati.

## 7 Il pacchetto etoolbox

Il pacchetto etoolbox (Lehman, 2011) è una preziosa "cassetta degli attrezzi" per *pdflatex*, *xelatex* e *lualatex*, tanto utile che spesso è caricato di default. Basta esaminare il file `.log` di una qualunque composizione recente, cercare la stringa `etoolbox` e constatare se il pacchetto in questione è stato caricato anche senza l'invocazione esplicita da parte dell'utente.

Questo pacchetto mette a disposizione dell'utente molte decine di nuovi comandi di cui almeno una cinquantina per gestire l'esecuzione condizionale di codice. Sarebbe troppo lungo elencarli e commentarli tutti in questo articolo, quindi rinvio il lettore alla documentazione del pacchetto. Ricordo che aprendo una finestra comandi (che prende nomi diversi con sistemi operativi diversi) e digitandovi dentro il comando

```
texdoc ⟨pacchetto⟩
```

si apre il visualizzatore di file PDF o il browser di rete, che permettono di leggere direttamente la documentazione di un qualsiasi ⟨*pacchetto*⟩. Talvolta il ⟨*pacchetto*⟩ dispone di diversi file di documentazione, per cui può essere utile aggiungere una opzione

```
texdoc -showall ⟨pacchetto⟩
```

In questo modo il programma di lettura della documentazione mostra l'elenco numerato di tutti i file di documentazione relativi al ⟨*pacchetto*⟩ specificato; rispondendo al prompt di sistema con il numero del file che interessa, quel file viene aperto e se ne può leggere il contenuto.

La documentazione di etoolbox è abbastanza vasta, ma è anche molto vasto il numero di comandi messi a disposizione; il paragrafo §3.5 nella pagina 12 comincia con i comandi di definizione e di gestione delle variabili booleane; il paragrafo §3.6 che comincia nella pagina 14, descrive una moltitudine di comandi condizionali, tanto che il paragrafo finisce nella pagina 23.

Tutti i comandi condizionali accettano diversi argomenti gli ultimi due dei quali specificano l'⟨*azione se vero*⟩ e l'⟨*azione se falso*⟩; in questo modo tutti i comandi sono piuttosto robusti e non ho mai riscontrato un caso in cui si "rompessero". Più avanti mostrerò un esempio in cui gli altri comandi condizionali o nativi di TeX o definiti dagli altri pacchetti si mostrano fragili, ma quelli di etoolbox sono robusti.

In un comando condizionale gli argomenti precedenti gli ultimi due hanno funzioni speciali che differiscono talmente da comando a comando che è necessario esaminarne la documentazione per comprenderne il significato e la sintassi.

Quello che balza agli occhi è la varietà di test che si possono fare per controllare lo stato dei *comandi* o dei *nomi dei comandi* (i comandi a cui è stato tolto il backslash che li introduce); per esempio:

```
\ifdef{⟨comando⟩}{⟨azione se vero⟩}%
    {⟨azione se falso⟩}
\ifcsdef{⟨nome del comando⟩}%
    {⟨azione se vero⟩}{⟨azione se falso⟩}
```

controllano entrambi se \⟨*comando*⟩ è definito, ma il primo fa riferimento al comando vero e proprio mentre il secondo controlla solo se esiste un comando con quel nome.

Le espressioni logiche con i vari connettori logici sono certamente eseguibili, ma forse la sintassi è più complessa di quella del pacchetto ifthen; secondo me è più chiara sebbene sia un poco più prolissa.

L'esempio di `\cleardoublepage` visto per il pacchetto ifthen diventa:

```
\newcommand*\cleardoublepage{\clearpage
  \ifboolexpr{bool{@twoside}
       and not
       test{\ifnumodd{\value{page}}}}%
    {\mbox{}\newpage
    \ifboolexpr{bool{@twocolumn}}
      {\mbox{}\newpage}{}%
    }{}
}
```

Nonostante il ripiegamento delle righe di codice si vede chiaramente che la sintassi del comando principale, che può tranquillamente venire annidato, è quella seguente:

```
\ifboolexpr{⟨espressione booleana⟩}%
    {⟨azione se vera⟩}{⟨azione se falsa⟩}
```

Si riconosce che i test definiti con `\newif` vengono usati come variabili booleane togliendo loro il prefisso `\if` ma vengono preceduti dal descrittore `bool`. Invece i test che usano argomenti vengono eseguiti con i comandi di etoolbox che prevedono l'inserimento dei due esiti negli ultimi due argomenti, saltando la specificazione di tali argomenti, ma premettendo il qualificatore `test` al comando condizionale con i suoi primi argomenti. I connettori logici `or`, `and` e `not` sono semplici 'parole', non comandi, e sono separate l'una dall'altra (e da `bool` e `test`) da semplici spazi; il comando complessivo non accetta righe vuote, ma permette di andare a capo rendendo più leggibile il codice.

Come si vede la sintassi è più articolata rispetto a quella del pacchetto ifthen ma è il prezzo pagato per ottenere una maggiore robustezza.

# 8 Confronto fra i vari comandi condizionali

Supponiamo di voler creare un pacchetto che fra le altre cose usi un test che per semplicità pensiamo creato con il comando `\newif`:

```
\newif\iftest
```

Nel definire un nuovo comando senza sovrapporlo a definizioni già esistenti, non possiamo ricorrere a un comando `\csprovideif` analogo al comando `\providecommand`. Dobbiamo quindi controllare se è già definito il test `\iftest`.

## 8.1 Comandi nativi di TEX

Creiamo il semplice file di prova seguente:

```
\documentclass{article}
   \newif\iftest
   \testfalse
%   \testtrue
\nofiles
\begin{document}
\ifdefined\iftest
   \typeout{definito}
\else
   \typeout{non definito}
\fi\fi
\end{document}
```

La documentazione di eTeX dice che `\ifdefined` non sviluppa il comando su cui esegue il test ma questo è solo parzialmente vero. Compilando il piccolo file di prova diverse volte, commentando via via sia la definizione di `\iftest` e i suoi comandi d'impostazione, sia solo l'uno o l'altro comando d'impostazione, se ne possono osservare gli effetti.

Nel caso in cui `\iftest` è definito e a seconda del suo stato, si scopre che ci vogliono uno o due comandi di fine del costrutto condizionale; quando ce n'è uno di troppo il programma di compilazione si arresta con l'errore: `extra \fi...`.

Il comando condizionale quindi è fragile e si è "rotto".

## 8.2 Comandi di ifthen

Si crei ora un altro file di prova come il seguente:

```
\documentclass{article}
   \newif\iftest
   \testfalse
%   \testtrue
\usepackage{ifthen}
\nofiles
\begin{document}
\ifthenelse{\isundefined{\iftest}}
  {\typeout{non definito}}
  {\typeout{definito}}
\end{document}
```

Si ripetano gli esperimenti svolti nel paragrafo precedente, commentando via via i comandi che si riferiscono al test `\iftest` nelle varie combinazioni; nuovamente si scoprirà che esiste almeno una combinazione in cui lo sviluppo del comando `\ifthenelse` si "rompe" e dà luogo ad un messaggio d'errore. Di nuovo questo comando, nonostante sia più robusto dei comandi nativi di TEX, non è robusto abbastanza.

## 8.3 Comandi di etoolbox

Si predisponga nuovamente di un piccolo file di prova come il seguente:

```
\documentclass{article}
   \newif\iftest
   \testfalse
%   \testtrue
\usepackage{etoolbox}
\nofiles
\begin{document}
\ifdef{\iftest}% oppure \ifcsdef{iftest}
  {\typeout{definito}}
  {\typeout{non definito}}
\end{document}
```

Per vederne gli effetti, si ripetano ora i soliti test con le varie combinazioni di righe commentate in merito al test `\iftest`; si cambi anche l'enunciato che contiene `\ifdef` con l'enunciato che contiene `\ifcsdef`; si scoprirà che in nessun caso il costrutto logico si "rompe"; non sono riuscito a creare altri esempi nei quali i comandi di etoolbox si dimostrino fragili.

## 8.4 Ancora sulla fragilità dei comandi condizionali

I comandi condizionali di etoolbox sono molto robusti perché chi ha scritto le macro ha sfruttato a fondo le estensioni di *eTeX*. In tal modo è riuscito a "detokenizzare" i comandi che a loro volta fossero argomenti di altre macro, facendoli così diventare delle stringhe di caratteri ed evitando che venissero eseguiti al momento sbagliato.

Evidentemente i comandi di ifthen, scritti nel 2001, assai prima di quando non siano state aggiunte le modifiche di ε-TEX nei vari motori di composizione, funzionano molto bene, ma non sono così robusti.

I comandi condizionali nativi di TEX si rivelano fragili perché vengono eseguiti troppo presto; secondo la metafora del tubo digerente di Knuth, essi vengono elaborati nella bocca, non nello stomaco, in modo che nello stomaco arrivi *solo* quella parte del ramo ⟨*azione se vero*⟩ o ⟨*azione se falso*⟩ da eseguire a seconda dell'esito del test. In questo modo, se l'argomento del comando primitivo `\ifdefined` è un comando condizionale, questo viene eseguito prima di vedere il resto della frase logica.

Quando l'esecuzione di uno dei tre programmi di composizione *pdflatex*, *xelatex* o *lualatex* si blocca per un errore di difficile interpretazione, molto spesso la causa è un comando condizionale primitivo eseguito al momento sbagliato. Il LATEX 3 Team sta facendo di tutto per eliminare tutti i comandi fragili, ma ancora il nucleo di LATEX, così come i molti pacchetti delle distribuzioni del sistema TEX che non siano stati aggiornati di recente, sono soggetti alla fragilità.

# 9 Definizione di alcuni comandi utili

## 9.1 Un salto pagina condizionale

Spesso componendo con LATEX si ottengono delle pagine con gli spazi verticali esageratamente allargati; quasi sempre tale effetto sarebbe causato da un comando di sezionamento che richiede almeno 4 o 5 righe di spazio verticale e che in mancanza di quest'ultimo fa proseguire la composizione alla pagina successiva. L'effetto è sgradevole e quindi sarebbe utile prima del comando di sezionamento poter dare un comando del tipo `\goodpagebreak` che esegua i controlli al nostro posto. Basta che tale comando confronti quanto spazio esiste ancora sulla pagina con un numero specificato di righe:

```
\newcommand*\goodpagebreak[1][4]{\par
\ifdimgreater
   {\dimexpr\pagegoal-\pagetotal}%
   {#1\baselineskip}%
   {\relax}{\newpage}}
```

Se sulla pagina è disponibile un numero di righe superiore al valore preimpostato o a quello specificato, allora il comando non esegue alcuna azione di sezionamento altrimenti esegue un fine pagina. Non voglio affermare che con i comandi nativi di TEX o i comandi del pacchetto ifthen non si sarebbe potuto ottenere un risultato robusto, ma passando attraverso i comandi di etoolbox siamo sicuri che questo costrutto è robusto.

## 9.2 Scalare un oggetto solo se eccede la giustezza

In questo esempio non faremo acrobazie di programmazione; cercheremo invece di confezionare un ambiente resizedisplay che scali alla giustezza corrente una figura o una tabella nel caso in cui le loro dimensioni naturali la eccedano.

Carichiamo il pacchetto graphicx e definiamo una nuova lunghezza e una nuova scatola in modo tale che il preambolo diventi:

```
\usepackage{graphicx}

\newlength\mydisplaywidth
\newbox{\mydisplay}
```

Poi definiamo il nuovo ambiente cominciando ad impostare la nostra nuova lunghezza al valore di `\columnwidth` o di `\textwidth` a seconda che si stia componendo a due o ad una colonna; non basta verificare la variabile booleana@twocolumn, perché la composizione a due colonne potrebbe essere eseguita dentro l'ambiente multicols senza che sia stata impostata la composizione a due colonne per l'intero documento. Ricordando che `\linewidth` rappresenta la giustezza corrente, cominciamo quindi a definire:

```
\newenvironment{resizedisplay}{%
\ifdimless{\linewidth}{\textwidth}%
{\setlength{\mydisplaywidth}{\columnwidth}}%
{\setlength{\mydisplaywidth}{\textwidth}}%
```

Siccome dobbiamo misurare la larghezza dell'oggetto da scalare, lo componiamo dentro l'ambiente lrbox, che metterà l'oggetto dentro la scatola `\mydisplay`; continuiamo quindi la definizione dei comandi di apertura del nostro ambiente con il codice:

```
\begin{lrbox}{\mydisplay}}
```

Ora l'ambiente è aperto e possiamo inserire la tabella (ambiente tabular) o la figura (`\includegraphics[...]{...}`). Inserito nell'ambiente quello che occorre, dobbiamo chiudere la scatola; in vista del fatto che l'ambiente potrebbe essere usato quando si è in modo verticale, dobbiamo eliminare l'eventuale rientro del capoverso. Perciò inseriamo subito:

```
{\end{lrbox}\noindent
```

Poi dobbiamo provvedere ai controlli sulle larghezze e provvedere eventualmente a scalare l'oggetto. L'oggetto però ora è inscatolato dentro la scatola `\mydisplay` perciò dobbiamo controllare la larghezza di questa scatola; senza ricorrere i comandi LATEX per misurare la larghezza degli oggetti (che verrebbero comunque messi dentro un'altra scatola per poter misurare le dimensioni di quest'ultima) usiamo i comandi nativi di TEX ed eseguiamo il controllo con i seguenti comandi:

```
\ifdimless{\wd\mydisplay}{\mydisplaywidth}%
   {\usebox{\mydisplay}}%
   {\resizebox{\mydisplaywidth}{!}%
      {\usebox{\mydisplay}}%
   }\ignorespaces
}
```

Il comando `\wd`, seguito dal nome della scatola, ne restituisce la larghezza (*width*). Il comando `\usebox` è il comando LaTeX per usare il contenuto della scatola il cui nome compare nel suo argomento[3].

Ora la sintassi per l'uso dell'ambiente è semplice:

```
\begin{resizedisplay}
⟨oggetto da scalare⟩
\end{resizedisplay}
```

ma non bisogna abusarne. Se l'oggetto è decisamente troppo grande, il rimpicciolimento potrebbe essere eccessivo, tale da renderne illeggibile il contenuto. Quando poi l'oggetto è costituito da una o più equazioni in display, l'uso di questo ambiente richiede alcune cautele, come per esempio quella di racchiudere l'espressione matematica dentro una minipage di cui bisogna determinare la larghezza "sperimentalmente" altrimenti si rischia di rimpicciolire troppo il contenuto matematico. La minipage consente di comporre equazioni in display, cosa che l'ambiente lrbox non consente, ma richiede anche una certa disponibilità a procedere per tentativi visto che essa richiede comunque che le si specifichi la giustezza.

L'esempio dimostra che definire nuovi comandi e nuovi ambienti è sempre utile per risparmiare la scrittura di codice relativamente complesso ma del quale è richiesta la stesura senza il minimo errore e nella giusta sequenza.

L'esempio non è particolarmente complesso e, invece dei comandi di etoolbox, si sarebbero potuti usare anche i comandi del pacchetto ifthen o i comandi condizionali nativi di TeX. Ma così facendo abbiamo la certezza della loro robustezza.

## 10 Conclusioni

Sono convinto che uno dei punti più delicati della programmazione di macro nei linguaggi che ci vengono offerti da TeX, LaTeX e dai vari pacchetti di estensione sia proprio costituito dai comandi condizionali.

I modi per scrivere comandi condizionali, da usare sia all'interno di file di classe o di pacchetti personali, ma anche dentro macro personali definite nel preambolo dei propri documenti, sono diversi e diversamente utili; più che altro essi differiscono per la fragilità o robustezza che questi comandi presentano nelle varie circostanze.

3. Per approfondire le informazioni relative a questi comandi si può consultare la guida tematica (Beccari, 2014a).

La fragilità è commentata e descritta anche nel documento di Gregorio (2009) di cui consiglio la lettura; capire la fragilità dei comandi condizionali è importante; sapere come fare per poter usare comandi meno fragili o, meglio ancora, decisamente robusti è ancora più importante.

## Riferimenti bibliografici


Beccari, C. (2014a). *Il LaTeX Reference Manual commentato*. GuIt. Scaricabile dal sito `www.guitex.org` dalla sezione Documentazione, sottosezione Guide Tematiche.

— (2014b). *Introduzione all'arte della composizione tipografica con LaTeX*. Gruppo utilizzatori Italiani di TeX e LaTeX. Scaricabile dal sito `www.guitex.org`, sezione Documentazione.

Bezos, J. (2013). *Babel*. TeX Users Group. La versione 3.9h è disponibile con ogni distribuzione aggiornata del sistema TeX completa e leggibile con `texdoc babel`.

Breitenlohner, P. (1998). *The ε-TeX Manual*. The NTS Team, TeX Users Group. Disponibile con ogni distribuzione del sistema TeX completa e leggibile con `texdoc etex`.

Carlisle, D. (2001). *The ifthen package*. TeX Users Group.

Gregorio, E. (2009). «Appunti di programmazione in TeX e LaTeX».

Knuth, D. E. (1996). *The TeXbook*. Addison Wesley, Reading, Mass., 16ª edizione.

Lamport, L. (1994). *LaTeX: A document preparation system – User guide and reference manual*. Addison Wesley, 2ª edizione.

Lehman, P. (2011). *The etoolbox package*. TeX Users Group. Disponibile con ogni distribuzione del sistema TeX completa e leggibile con `texdoc etoolbox`.

Noirel, J. (2009). *The xifthen package*. Disponibile con ogni distribuzione del sistema TeX completa e leggibile con `texdoc xifthen`.



▷ Claudio Beccari
Villarbasse
`claudio dot beccari at gmail.com`