The dirtree package
Directory Tree

Jean-Côme Charpentier*

Version 0.31 2012/11/02
Documentation revised November 2, 2012

Abstract

Package dirtree allows to display directory tree, like in the windows explorator.

Contents

1 Introduction

During a discussion on fctt (`fr.comp.text.tex`) about directory tree and how display such a structure, it appeared that there wasn't many packages which do the job.

One obvious solution is to use PSTricks but some people don't like or don't know this package, so I made the first release of dirtree.

In fact, I didn't plan to send it in CTAN but Robin Fairbairns and Danie was very convincing!

*Jean-Come.Charpentier@wanadoo.fr

2 Usage

Package dirtree works both on Plain TEX and LATEX. No surprise to call it:

```
\usepackage{dirtree}
```

for LATEX and

```
\input dirtree
```

for Plain TEX.

Since version 0.3, dirtree has some package options. We'll see these options one by one.

`\dirtree` The main macro is `\dirtree` which take one argument (the tree structure). This tree structure is a sequence of

```
.<level><space><text node>.<space>
```

Note that there is a dot in the beginning and another one at the end of each node specification. The spaces are very important: if you forgot the space before the `level` there will be an error and if you forgot the space after the last dot, you don't indicate the end of the node. Since an end of line is like a space for TEX, I recommand to write a node per line in the source file: it's handy and more readeable.

The `level` indicates the node depth in the tree. There is two rules you must respect:

1. The root must have the level one.
2. When you create a node, if the last node have the level n, the created node must have a level between 2 and $n + 1$.

In fact, you can indicates a level greater than $n + 1$ if one node have a level n somewhere in the tree but the result will be strange!

A node of level n will be connected to the last node defined which has a level lesser or equal to n.

For example, the code

```
\dirtree{%
 .1 /.
 .2 bin.
 .2 home.
 .3 jeancome.
 .4 texmf.
 .5 tex.
 .6 latex.
 .7 dirtree.
 .3 jeancomeson.
```

```
  .3 jeancomedaughter.
  .2 usr.
  .3 bin.
  .3 games.
  .4 fortunes.
  .3 include.
  .3 local.
  .4 bin.
  .4 share.
  .5 texmf.
  .6 fonts.
  .6 metapost.
  .6 tex.
  .3 share.
 }
```

give the result

```
/
├─bin
├─home
│ ├─jeancome
│ │ └─texmf
│ │   └─tex
│ │     └─latex
│ │       └─dirtree
│ ├─jeancomeson
│ └─jeancomedaughter
└─usr
  ├─bin
  ├─games
  │ └─fortunes
  ├─include
  ├─local
  │ ├─bin
  │ └─share
  │   └─texmf
  │     ├─fonts
  │     ├─metapost
  │     └─tex
  └─share
```

Note the `%` after the left brace in the beginning: it's important because the first character encountered must be a dot.

`\DTstyle` A text node is typeset with the command `\DTstyle`. Its default value is `\ttfamily` when you are under LaTeX and `\tt` when you are under Plain TeX. You can redefine this macro as you want, it is used with the syntax `{\DTstyle{text node}}`, so you can use both `\ttfamily` and `\texttt` for ex-

ample.

`\DTcomment` The `\DTcomment` command allows to put text at the right side, with leaders. The syntax is

```
\DTcomment{comment text}
```

`\DTstylecomment` The style of comment is defined by `\DTstylecomment`. Its default value is `\rmfamilly` under LaTeX and `\rm` under Plain TeX, and it acts like `\DTstyle`. Here is an example: the code

```
\renewcommand*\DTstylecomment{\rmfamily\color{green}\textsc}
\renewcommand*\DTstyle{\ttfamily\textcolor{red}}
\dirtree{%
 .1 /.
 .2 bin.
 .2 home.
 .3 jeancome.
 .4 texmf.
 .5 tex.
 .3 jeancomeson\DTcomment{Guillaume}.
 .3 jeancomedaughter\DTcomment{Mathilde}.
 .2 usr.
 .3 bin.
}
```

give the result

In this example we have used the xcolor package.

You can build complex text node. For example, the code

```
\dirtree{%
 .1 /.
 .2 bin \ldots{} \begin{minipage}[t]{5cm}
                  This directory holds executable files (binary
                  files or link on binary files){.}
       \end{minipage}.
 .2 home \ldots{} \begin{minipage}[t]{5cm}
                   jeancome\\
                   guillaume\\
```

```
                    mathilde\\
                  \end{minipage}.
  .4 texmf.
 }
```

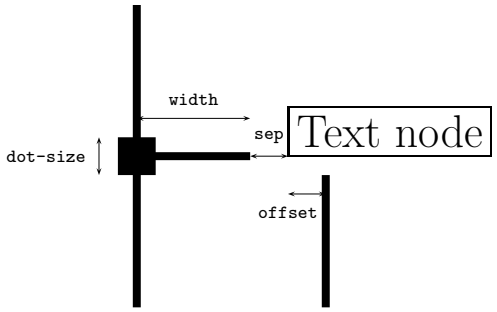
give the result

```
/
├─bin ...  This directory holds
│          executable files (binary
│          files or link on binary
│          files).
├─home ...  jeancome
│           guillaume
│           mathilde
│
└─texmf
```

We don't encourage to try too complicated code. Package dirtree is still fragile! Note that we pay attention to use optional parameter `[t]` in order to have a right vertical alignment with horizontal rules.

`\DTsetlength` Some dimensions can be changed using the `\DTsetlength` command. The syntax is:

```
\DTsetlength{offset}{width}{sep}{rule-width}{dot-size}
```



The default value are:

- `offset = 0.2em`
- `width = 1em`
- `sep = 0.2em`
- `rule-width = 0.4pt`
- `dot-size = 1.6pt`

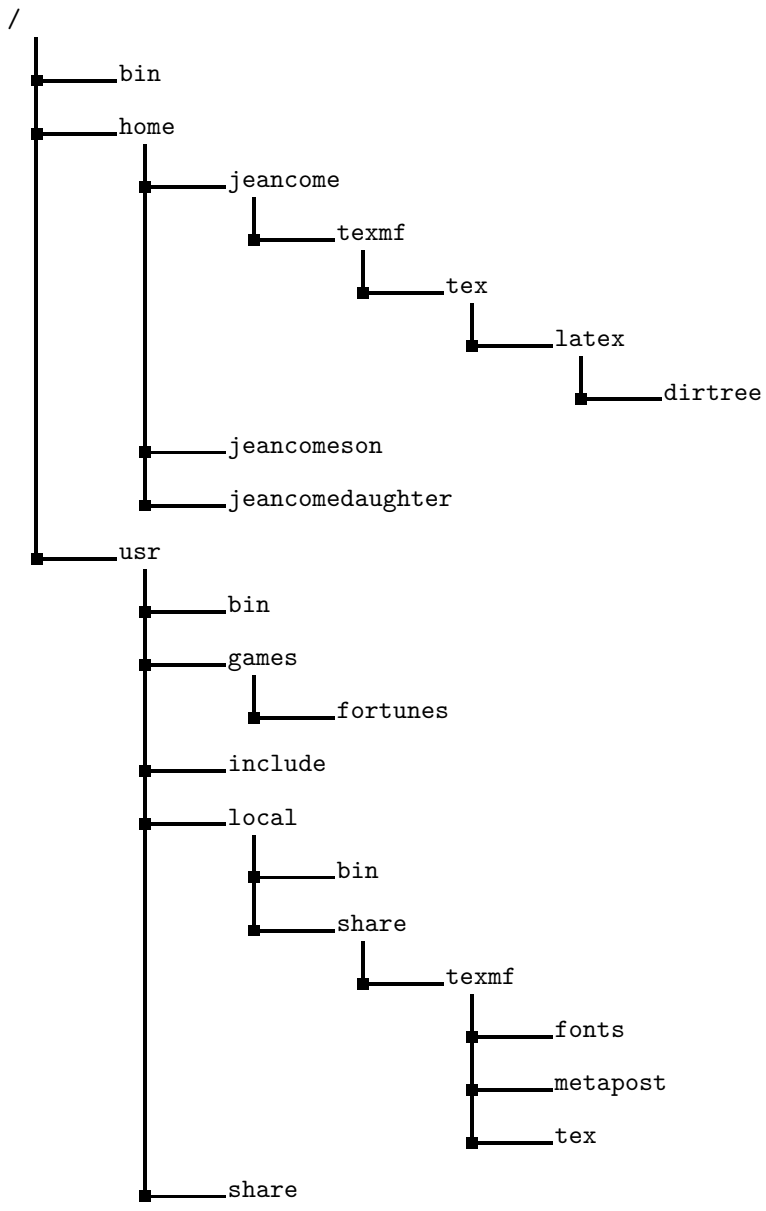
`\DTbaselineskip` The last length parameter is `\DTbaselineskip` which indicates the skip be-

tween lines of the tree.

If we typeset the first example with

```
\setlength{\DTbaselineskip}{20pt}
\DTsetlength{1em}{3em}{0.1em}{1pt}{4pt}
```

we obtain the (strange) result:



Note that dirtree package is not able to split tree on several pages. If this case occurs, the result will be very strange with overfull rules. I suppose that the best is to place such trees inside floats.

3 ToDo

- Parameters with xkeyval syntax;
- Command `\DTsplittree` to allows a tree to be typeseted on several pages;
- Style parameters to rules (color for example) and gap between text and comment (by now it's `\dotfill`).
- Dimension parameter `abovetreeskip` and `belowtreeskip`.

<*latex-wrapper>

4 dirtree LaTeX Wrapper

Nothing special here but the `\DT@fromsty` definition. This latter is intended to check if dirtree is called under LaTeX (with `\usepackage`) or under Plain TeX.

```
\NeedsTeXFormat{LaTeX2e}[1995/06/01]
\ProvidesPackage{dirtree}[\filedate\space v\fileversion\space
                          package wrapper for dirtree]
\newcommand*\DT@fromsty{}
\input{dirtree.tex}
\ProvidesFile{dirtree.tex}
  [\filedate\space v\fileversion\space 'dirtree' (jcc)]
```

</latex-wrapper> <*tex>

5 dirtree Code

An "hello" message.

```
\message{‘dirtree’ v\fileversion, \filedate\space (jcc)}
```

Save at current catcode and make @ a letter

```
\edef\DTAtCode{\the\catcode‘\@}
\catcode‘\@=11
```

Define `\LOOP`, `\REPEAT`, and `\ITERATE` like `\loop`, `\repeat`, and `\iterate`. The uppercase form allows to place loop inside loop.

```
\long\def\LOOP#1\REPEAT{%
  \def\ITERATE{#1\relax\expandafter\ITERATE\fi}%
  \ITERATE
  \let\ITERATE\relax
}
\let\REPEAT=\fi
```

Define some LaTeX macros if we work under Plain TeX. `\@namedef`-like for `\edef`.

```
\expandafter\ifx\csname DT@fromsty\endcsname\relax
  \def\@namedef#1{\expandafter\def\csname #1\endcsname}
  \def\@nameuse#1{\csname #1\endcsname}
  \long\def\@gobble#1{}
\fi
\def\@nameedef#1{\expandafter\edef\csname #1\endcsname}
```

Offset between vertical rule below text and text left boundary.

```
\newdimen\DT@offset \DT@offset=0.2em
```

Length of horizontal rule.

```
\newdimen\DT@width \DT@width=1em
```

Gap between horizontal rule and text.

```
\newdimen\DT@sep \DT@sep=0.2em
```

`\DT@offset` + `\DT@width` + `\DT@sep`

```
\newdimen\DT@all
\DT@all=\DT@offset
\advance\DT@all \DT@width
\advance\DT@all \DT@sep
```

Rule thickness

```
\newdimen\DT@rulewidth \DT@rulewidth=0.4pt
```

Size of square junction.

```
\newdimen\DT@dotwidth \DT@dotwidth=1.6pt
```

baselineskip inside tree.

```
\newdimen\DTbaselineskip \DTbaselineskip=\baselineskip
```

Max index node.

```
\newcount\DT@counti
```

Current index node

```
\newcount\DT@countii
```

`\DT@countiii` = `\DT@countii` − 1. That is, Previous index node.

```
\newcount\DT@countiii
```

Last node of a level lesser or equal to current one.

```
\newcount\DT@countiv
```

`\DTsetlength` `\DTsetlength` allows to define dimensions in use for the directory tree (see above).

```
\def\DTsetlength#1#2#3#4#5{%
  \DT@offset=#1\relax
  \DT@width=#2\relax
  \DT@sep=#3\relax
```

`\DT@all` is the width of a whole column.

```
  \DT@all=\DT@offset
  \advance\DT@all by\DT@width
  \advance\DT@all by\DT@sep
  \DT@rulewidth=#4\relax
  \DT@dotwidth=#5\relax
}
```

`\DTstyle` is the style used to typeset nodes. `\DTstylecomment` is the style used to typeset comments. Since TEX and LATEX are very different, we test the format used before initializations.

`\DTstyle`
`\DTstylecomment`

```
\expandafter\ifx\csname DT@fromsty\endcsname\relax
  \def\DTstyle{\tt}
  \def\DTstylecomment{\rm}
\else
  \def\DTstyle{\ttfamily}
  \def\DTstylecomment{\rmfamily}
\fi
```

`\DTcomment` `\DTcomment` places comment in a line of the tree.

```
\def\DTcomment#1{%
  \kern\parindent\dotfill
  {\DTstylecomment{#1}}%
}
```

In order to save some lengths we create newdimen

```
\newdimen\DT@indent
\newdimen\DT@parskip
\newdimen\DT@baselineskip
```

`\dirtree` `\dirtree` is the main package macro.

```
\def\dirtree#1{%
```

Change some parameters (save them before).

```
\DT@indent=\parindent
\parindent=\z@
\DT@parskip=\parskip
\parskip=\z@
\DT@baselineskip=\baselineskip
\baselineskip=\DTbaselineskip
\let\DT@strut=\strut
\def\strut{\vrule width\z@ height0.7\baselineskip depth0.3\baselineskip}%
```

Read the argument and before that, initialize counters. `\DTcounti` is the current index node.

```
\DT@counti=\z@
\let\next\DT@readarg
\next#1\@nil
```

When `\DT@readarg` has done its job, the node levels and the node texts are saved in `\DT@level@<index>` and `\DT@body@<index>` respectively. `\DT@counti` holds the greater index. We can now display the tree.

Firstly, display the root. For that, the text is boxed.

```
\dimen\z@=\hsize
\advance\dimen\z@ -\DT@offset
\advance\dimen\z@ -\DT@width
\setbox\z@=\hbox to\dimen\z@{%
  \hsize=\dimen\z@
  \vbox{\@nameuse{DT@body@1}}%
}%
```

We change the height and the depth of this box in order to have the same total height and a height of `0.7\baselineskip`, that is, the height of `\strut`.

```
\dimen\z@=\ht\z@
\advance\dimen0 by\dp\z@
\advance\dimen0 by-0.7\baselineskip
\ht\z@=0.7\baselineskip
\dp\z@=\dimen\z@
```

Then we display this box with an indentation as if there had a level 0.

```
\par\leavevmode
\kern\DT@offset
\kern\DT@width
\box\z@
\endgraf
```

Initialize index for the loop.

```
\DT@countii=\@ne
\DT@countiii=\z@
```

`\dimen3` holds the height of the node in the tree. In fact, the bottom of the node since this dimension is used to connect vertical rules.

```
\dimen3=\dimen\z@
```

`\DT@lastlevel@<level>` holds the baseline of the last node in level `<level>`.

```
\@namedef{DT@lastlevel@1}{-0.7\baselineskip}%
```

Loop for displaying the remainder of the tree.

```
\loop
```

Exit loop when the last current index is lesser or equal to max index.

```
\ifnum\DT@countii<\DT@counti
```

`\DT@counti` holds current index and `\DT@countii` holds previous index (just current index minus one).

```
  \advance\DT@countii \@ne
  \advance\DT@countiii \@ne
```

Horizontal offset for the text:
(`current level` -1) $\times$ `DT@all` + `DT@offset`.

```
  \dimen\z@=\@nameuse{DT@level@\the\DT@countii}\DT@all
  \advance\dimen\z@ by\DT@offset
  \advance\dimen\z@ by-\DT@all
  \leavevmode
  \kern\dimen\z@
```

Look for last node in previous level in order to know how connect the current node.

```
  \DT@countiv=\DT@countii
  \count@=\z@
  \LOOP
```

Look for previous node

```
    \advance\DT@countiv \m@ne
```

Repeat until this previous node has a level lesser or equal to current level.

```
    \ifnum\@nameuse{DT@level@\the\DT@countiv} >
      \@nameuse{DT@level@\the\DT@countii}\relax
    \else
      \count@=\@ne
    \fi
  \ifnum\count@=\z@
  \REPEAT
```

Now `\DT@countiv` holds the index node connected to current node.
We box the text node.

```
  \edef\DT@hsize{\the\hsize}%
  \count@=\@nameuse{DT@level@\the\DT@countii}\relax
```

Since text node is vboxed, we use a `\hsize` minus horizontal current offset.

```
  \dimen\z@=\count@\DT@all
  \advance\hsize by-\dimen\z@
  \setbox\z@=\vbox{\@nameuse{DT@body@\the\DT@countii}}%
```

Restore `\hsize`.

```
  \hsize=\DT@hsize
```

Change height and depth in such a way that height is `0.7\DT@baselineskip` (that is, the `\strut` height), and total height is unchanged.

```
    \dimen\z@=\ht\z@
    \advance\dimen\z@ by\dp\z@
    \advance\dimen\z@ by-0.7\baselineskip
    \ht\z@=0.7\baselineskip
    \dp\z@=\dimen\z@
```

Save the height of the box in tree. The last node is the last node in its level!

```
    \@nameedef{DT@lastlevel@\the\DT@countii}{\the\dimen3}%
```

`\dimen3` holds the vertical position of the bottom.

```
    \advance\dimen3 by\dimen\z@
    \advance\dimen3 by0.7\baselineskip
```

Display vertical rule

```
    \dimen\z@=\@nameuse{DT@lastlevel@\the\DT@countii}\relax
    \advance\dimen\z@ by-\@nameuse{DT@lastlevel@\the\DT@countiv}\relax
    \advance\dimen\z@ by0.3\baselineskip
```

If this vertical rule connect two nodes which have different level, the rule must be reduced by `0.5\baselineskip` (the rule don't rise up to the baselineskip of the connected node).

```
    \ifnum\@nameuse{DT@level@\the\DT@countiv} <
        \@nameuse{DT@level@\the\DT@countii}\relax
      \advance\dimen\z@ by-0.5\baselineskip
    \fi
```

Display vertical rule

```
    \kern-0.5\DT@rulewidth
    \hbox{\vbox to\z@{\vss\hrule width\DT@rulewidth height\dimen\z@}}%
    \kern-0.5\DT@rulewidth
```

Display square junction.

```
    \kern-0.5\DT@dotwidth
    \vrule width\DT@dotwidth height0.5\DT@dotwidth depth0.5\DT@dotwidth
    \kern-0.5\DT@dotwidth
```

Display horizontal rule and gap between horizontal rule and text node.

```
    \vrule width\DT@width height0.5\DT@rulewidth depth0.5\DT@rulewidth
    \kern\DT@sep
```

Display text node.

```
    \box\z@
```

New paragraph for next node.

```
    \endgraf
  \repeat
```

Restore indentation, paragraph skip, and `\strut`.

```
  \parindent=\DT@indent
  \parskip=\DT@parskip
  \DT@baselineskip=\baselineskip
```

```
149   \let\strut\DT@strut
150 }
```

`\DT@readarg` The first processing step is to read the whole tree. It's a recursive macro: each call process one node, that is, a

```
.<index> <text node>.<space>
```

in the source file.

```
151 \def\DT@readarg.#1 #2. #3\@nil{%
```

`\DT@counti` is the current index.

```
152   \advance\DT@counti \@ne
```

save level node in `\DT@level@<index>` and text node in `\DT@body@<index>`. Two dirtree `\strut` are added to text node in order to ensure a right vertical alignment, according to dirtree `\baselineskip`

```
153   \@namedef{DT@level@\the\DT@counti}{#1}%
154   \@namedef{DT@body@\the\DT@counti}{\strut{\DTstyle{#2}\strut}}%
```

If `#3` is not empty, it contains the remainder nodes and macro calls itself. Otherwise, recursive call is stopped.

```
155   \ifx\relax#3\relax
156     \let\next\@gobble
157   \fi
158   \next#3\@nil
159 }
```

Restore at catcode.

```
160 \catcode`\@=\DTAtCode\relax
```

</tex>

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.