xparse: beyond `\newcommand`

Joseph Wright

14th November 2009

\newcommand

```
\newcommand\foo{Code with no arguments}

\newcommand\foo[2]{Code using #1 and #2}

\newcommand\foo[2][]{Code using #1 and #2}

\newcommand\foo[2][default]
  {Code using #1 and #2}
```

\newcommand*

```
\newcommand*\foo{Code with no arguments}

\newcommand*\foo[2]{Code using #1 and #2}

\newcommand*\foo[2][]{Code using #1 and #2}

\newcommand*\foo[2][default]
  {Code using #1 and #2}
```

The aims

- Intersperse mandatory and optional arguments
- More types of argument without code
- Mix long and short arguments
- Create engine robust commands
- Single consistent syntax
- Define all arguments in one place

\...DocumentCommand

- \NewDocumentCommand
- \RenewDocumentCommand
- \ProvideDocumentCommand
- \DeclareDocumentCommand

\...DocumentCommand

- \NewDocumentCommand
- \RenewDocumentCommand
- \ProvideDocumentCommand
- \DeclareDocumentCommand

Syntax

\DeclareDocumentCommand \⟨*name*⟩ {⟨*arg. spec.*⟩} {⟨*code*⟩}

Mandatory arguments

```
\DeclareDocumentCommand \foo { }
  {Code using no arguments}

\DeclareDocumentCommand \foo { m }
  {Code using #1}

\DeclareDocumentCommand \foo { m m }
  {Code using #1 and #2}

\DeclareDocumentCommand \foo { m m m }
  {Code using #1, #2 and #3}
```

Mandatory arguments

Mixing short and long

```
\DeclareDocumentCommand \foo { m m m }
  {Three short arguments}

\DeclareDocumentCommand \foo { +m +m +m }
  {Three long arguments}

\DeclareDocumentCommand \foo { m +m m }
  {Only #2 is long}
```

Optional arguments

Like LaTeX 2ε

```
\DeclareDocumentCommand \foo { O{} }
  {One optional argument}

\DeclareDocumentCommand \foo { O{} O{} m }
  {Two optionals then a mandatory}

\DeclareDocumentCommand \foo { O{default} m }
  {First argument optional with default value}
```

Optional arguments

Beyond LaTeX 2ε

```
\DeclareDocumentCommand \foo { o m }
  {%
    \IfNoValueTF {#1}
      {Code for just #2}
      {Code for #1 and #2}%
  }
```

Optional arguments

Beyond LaTeX 2ε

```
\DeclareDocumentCommand \foo { o m }
  {%
    \IfNoValueTF {#1}
      {Code for just #2}
      {Code for #1 and #2}%
  }
```

```
\DeclareDocumentCommand \foo { o m }
  {%
    \edef\something{%
      \IfNoValueF {#1} {#1}%
    }%
    Something with #2
  }
```

Optional arguments


```
\newcommand\foo[2][]{%
  % Code
}

\foo[\baz[arg1]]{arg2}

#1 = \baz[arg1
#2 = ]
```

Optional arguments


```
\DeclareDocumentCommand \foo { O{} m } {%
  % Code
}

\foo[\baz[arg1]]{arg2}

#1 = \baz[arg1]
#2 = arg2
```

Stars

```
\DeclareDocumentCommand \foo { s m }
  {%
    \IfBooleanTF #1
      {Process #2 with a star}
      {Process #2 without a star}%
  }
```

\newcommand itself

```
\DeclareDocumentCommand \newcommand
  { s +m O{0} +o +m }
  {%
     % #1 = Star: \BooleanTrue or \BooleanFalse
     % #2 = Command name
     % #3 = Number of arguments
     % #4 = Default for first argument if
     %       optional or \NoValue otherwise
     % #5 = Code
  }
```

Generalised optional types

```
\DeclareDocumentCommand \foo { d<> m }
  {First optional argument in angle brackets}

\DeclareDocumentCommand \foo { D(){default} m }
  {First optional argument in brackets}

\DeclareDocumentCommand \foo { t+ s m }
  {Optional +, then optional *, then mandatory}
```

Mandatory arguments beyond `m`

```
\DeclareDocumentCommand \foo { u{stop} }
  {Reads everything up to ‘stop’}

\DeclareDocumentCommand \foo { l }
  {
    Reads up to opening brace:
    like \def#1#{...}
  }
```

Environments

- Set of four commands:
  - `\NewDocumentEnvironment`
  - `\RenewDocumentEnvironment`
  - `\ProvideDocumentEnvironment`
  - `\DeclareDocumentEnvironment`
- Same argument system as for document commands
- Arguments available at beginning and end of environment
- `\`⟨*environment*⟩ and `\end`⟨*environment*⟩ defined for LaTeX 2$_\varepsilon$ (will not be used for LaTeX3)

Fully expandable commands

```
\DeclareExpandableDocumentCommand \foo
  { o m } {Expandable code using #1 and #2}
```

- *Exceptional* circumstances only!
- Some restrictions, such as final mandatory argument
- xparse enforces restrictions

Summary

- Flexible mechanism to define commands
- Clear syntax helps to ‘self document’
- Naturally robust commands created
- Ready for use now: on CTAN and stable