

## 1 Tikz (visual ref.)

General invocations:

```
\begin{tikzpicture}{} [<option>*]]{...}
\tcbset{<option>*}
```

```
\begin{tikzpicture}[...]{...} % start envt
\tikz{...} % inline diagram
\tcbset{...} %  $\Delta$  settings mid-course
```

Possible options are many, including:

- remember picture
- overlay
- $[x|y] = (\text{coord})$
- execute at end picture
- test

### Styles & Aliases

```
\begin{tikzpicture}{}
[<alias>/ .style={<style>*}]
```

Possible styles include:

style	example val
inner sep	2pt
text	green
font	small
anchor	west
align	left
pos	.5
decorate	–
decoration	snake
fill	green
color	green
draw	green
rounded corners	.2pt

Styles can be appended to or parameterized:

```
[mystyle/.style={draw=#1,fill=#1!50}]
[mystyle/.append style=blue!50]
```

### Organization

```
\begin{scope}[<option>*] % sub-envmt
\scoped[on background layer] % one-liner
```

### Data & Control

Wrap the following in `\tikzmath{...}` :

```
\a = 4*5+6; % assignment
if \k>170 then {let \c = blue;} % conditional
for \x in {...} % looping
func product(\x,\y){return \x*\y;;} % func
```

Low-level parsing is similar to  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ :

```
\def\myarray{{1,"two",2+1,sin(\i*5)}} % array
\pgfparse {\myarray[3]} % extract from
\foreach \i in{0,...,6} % loop over
\pgfmathparse{sqrt(10)} % apply func
\pgfmathparse{\pi} % constant  $\pi$ 
\pgfmathresult % return result
```

$\exists$  many built-in math functions:

abs	divide	less	rad
acos	e	ln	rand
add	equal	log10	real
and	factorial	log2	rnd
array	false	max	round
asin	floor	min	scalar
atan	frac	mod	sec
atan2	gcd	multiply	sign
bin	greater	neg	sin
ceil	height	not	sinh
cos	[H h]ex	notequal	sqrt
cosec	int	notgreater	subtract
cosh	ifthenls	notless	tan
cot	iseven	oct	tanh
deg	isodd	or	true
depth	isprime	pi	veclen
div		pow	width

## 2 Coordinates

```
([<option>*] <coordspec>)
```

Position and name a new coordinate:

```
\coordinate(mycoord) at(1,1);
\draw(mycoord)--(2,2) coordinate(mycoord2);
```

Ref existing or anonymous, on-the-fly coords:

type	example
named	(my_coord)
anchor	(my_node.north east)
absolute	(1,-3)
polar	(30:2cm)
relative	++(2,2)
non-updating	+(2,2)
rotational	([turn]-45:1cm)
factor-tween	(node_a)!.25!(node_b)
abs-tween	(node_a)!.1cm!(node_b)
math	\$(my_name) + (1,3)\$
intersect	(<coor_a> -  <coord_b>)

### Coordinate Systems

Above examples make “implicit” reference to desired “coordinate system.” To do so explicitly:

```
(<csname> cs: <coordspec>)
```

where `<csname>`  $\in$ : canvas, xyz, canvas polar, barycentric, node, or tangent.

Layer a new (scoped) canvas coordinate system:

```
\begin{scope}[xshift=6cm] ...
```

Use multiple cs’s with arbitrary complexity:

```
\draw[](A) -- ( $(A)!.85!(B)$ ); % tween+node
\node at($(A)!.5!(B)$) -| $(C)!.5!(D)$) % 3cs's
```

### 3 Dimensional Coordinates

```
(xyz cylindrical cs: radius=1) % cylindrical...
(xyz cylindrical cs: angle=90) % ...polar
(xyz cylindrical cs: z=1cm) % z-dir
(xyz spherical cs:radius=1,lat.=90) % sphér.
[canvas is xy plane at z] % draw to 2d plane
```

## 3 Path

```
\path[<option>*]
(<from>) -- (<to>) [-{<to>}- (<to>)]*;
```

(where `<from>`, `<to>` are coords ... see below)

Generally, paths start with a “move-to” operation (indicated by unpreceded coord), followed by various “path-to” operations. Paths can be drawn, filled, clipped, patterned, and shaded using `\path[<action>]`, or with shortcuts to the same like `\filldraw` or `\shade` Some basic examples:

```
\path[draw] (0,0) -- (5,2); % a line
\filldraw (-1,5) arc(180:120:1); % filled arc
\shadedraw (210:19mm); % using polar
```

### Path Operations

$\exists$  various “path operations,” exemplified below:

```
\draw (5,0) -- (6,1) -- (6,0) -- cycle; % line-to
\draw (a.north) |- (b.west); % line-to
\draw (0,0) -- (2,0) .. % crve-to
controls (1,1) and (2,2) .. cycle; % crve-to
\draw (.5,1) rectangle (2,0.5); % rect.
\draw (1,0) circle [radius=5mm]; % circle
\draw (1,0) circle % ellipse
[x radius=1cm, y radius=5mm]; % ellipse
\draw (8,0) arc [start angle=0, % arc
end angle=270, x radius=1cm, % arc
y radius=5mm] -- cycle; % arc
\draw (0,0) grid [step=.75cm] (3,2);% grid
\draw (0,0) parabola % parab.
bend (.75,1.75) (1,1.5); % alt 1
[bend pos=0.5] bend +(0,2) +(3,0); % alt 2
\draw (0,0) sin (1,1) cos (2,0); % sin
\draw svg [M 0 0 L 20 20 h 10 a 10]; % svg
\draw plot coord's {(a) (b) (c)}; % plot
\draw (a) to [out=135,in=45] (b); % to-path
(Labelling to-paths allow them to be styled.)
\draw (0,0) foreach \x in {1,...,3} % foreach
{- (\x,1) -- (\x,0) }; % foreach
\draw let \p{foo} = (1,1), \p2 = (2,0)% let
in (0,0) -- (\p2) -- (\p {foo}); % let
\draw (0,0) to[out=90,in=180] % node
node [sloped,above] x (3,2); % node
\draw (1,1) -- (2,2) pic [seagull]; % pic
(pics can be coded, path-positioned, & animated)
\draw :xshift = % animt'n
{0s="0cm", 30s="-3cm", repeats} % ""
(0,0) circle (5mm); % ""
```

### General Path Options

name	rounded corners
every path	sharp corners
insert path	color
rounding corners	help lines

### Draw Options

line width	densely dotted
ultra thin	loosely dotted
very thin	dashed
semithick	densely dashed
thick	loosely dashed
very thick	dash dot
ultra thick	densely dash dot
line cap	loosely dash dot
line join	dash dot dot
dash pattern	densely dash dot dot
dash phase	loosely dash dot dot
dash	double
dash expand off	double distance
solid	""betw. line centers
dotted	dbl eql sign distance

### Miscellaneous Path Actions

$\exists$  a number of fill and shade options in addition to fill “patterns” and “shading” types:

```
\pattern[pattern color=white, %
pattern=bricks] ... % pattern
\shadedraw [shading=axis] ... % shade type
fill % even odd rule
pattern % path picture
pattern color % shading
nonzero rule % shading angle
```

Manage picture-text interaction by controlling “bounding box” with trim,trim left,trim right, or like:

```
\draw[use as bounding box] ...
```

Clip around any kind of path (drawn, filled, etc):

```
\draw[clip] (0,0) circle (1cm)
```

### Plot Path Operation

A simplified, tikz-native alternative to pgfplots.

```
\draw plot coordinates % plot on a path
{ (0,0) (1,2) (3,0) ... } % simple coords
\draw plot[mark=x,smooth] % options
file {folder/file.table} % extern data
\draw plot (\x, {\sin(\x r)}) % a function
\draw [domain=-3:3,variable=\t] % paramet.
plot({\sin(\t r)},{cos(\t r)}) % circle
```

Options:

variable	mark repeat	const plot
samples	mark phase	jump mark
domain	mark indices	$[x y]$ comb
samples at	mark size	polar comb
$[x y]$ range	smooth	$[x y]$ bar
mark	tension	

## 4 Arrows & Arrow-Heads

```
\path[<begin_hd_spec>?-<end_hd_spec>?]
\path[-{Latex[... ]}] % long form
\path[<->] % shorthand
```

For consistency, redefine `>` head as desired shape within document as a whole or limited to one `\begin{scope}`:

```
\tikz [<-> /.tip = Stealth]
```

`begin_head_spec` and `end_head_spec` can include multiple heads or caps, can be separated or not, and can connected by lines or not, eg:

<code>\draw [-&gt;[sep=1pt]&gt;]]</code>	% separated
<code>\draw [-&gt;_&gt;]</code>	% ibid
<code>\draw [-&gt;&gt;[sep=2pt]]</code>	% space after
<code>\draw [&lt;.&lt;&lt;-&gt;.&gt;&gt;]</code>	% halt lines

#### Arrow Sizing, Positioning Options

width	scale	sep
width'	scale length	color
inset	scale width	fill
inset'	arc	open
angle	slant	shorten
angle'	reversed	shorten by
length	swap	

#### Heads & Caps

With following options, most of the following can be reversed, opened, or slanted.

arc barb	tee barb	latex
bar	implies	rectangle
bracket	to	square
hooks	circle	stealth
parenthesis	diamond	triangle
straight barb	ellipse	turned square
	kite	

Line “cap”s are an alternative to arrow “head”s:

butt cap	triangle cap
round cap	fast round
	fast triangle

## 5 Nodes & Edges

<code>\node [&lt;setting&gt;*]</code>
<code>at (&lt;coord&gt;)?</code>
<code>(&lt;name&gt;?) {&lt;text&gt;?}</code>

#### Options

setting	example
anchor	west
align	right justify
text width	.85cm
pos	.5
below of	my_node
text depth	5.7cm
label	{xshift=.5mm}
shape	rectangle callout

#### Matrices of Nodes

Create a set of nodes with a **matrix**:

<code>\matrix [matrix of nodes]{</code>	% create
<code>1 &amp; 2 &amp; 3 \\</code>	% basic syntax
<code>4 &amp; 5 &amp;  red  6 }</code>	% pass node opt'n

Other matrix options ∈:

draw	row sep	row # column #
anchor	every odd col.	above delimiter
column sep	every even col.	left delimiter

#### Edges

<code>\path [&lt;setting&gt;*]</code>
<code>(&lt;from&gt;) edge [&lt;setting&gt;*] (&lt;to&gt;);</code>

... where settings ∈:

out	bend right	<-
in	->	-[<a_head>[opt*]]

## 6 Graphs

Edges connect nodes or groups of nodes. Edges, nodes, groups, and graphs all have options.

<code>\graph [&lt;graph_opt ’n&gt;*] {&lt;graph_spec ’n&gt;}</code>
---

<code>\graph{ a -&gt; { b, c } -&gt; d }</code>	% basic examp.
---	----------------

#### Graph-level Options

nodes	simple	name separator
edges	multi	typeset
edge node	grow[up down]	empty nodes
edge label	grow[left right]	trie
edge quotes	branch[up down]	layered layout
left anchor	branch[left right]	quick
fresh nodes		color class
<code>\graph [grow down]</code>		% placement...
<code>\graph [branch right]</code>		% ... options

#### Graph Specifications

Graph specification (within `\graph{...}` ) comprise a list of `<group_spec>` s, each of which comprises a list of `<chain_spec>` s, each of which contains a list of nodes separated by edges.

<code>x1/\$x_1\$ -&gt; x2 [as=\$x_2\$, red]</code>	% naming
<code>b--[thick] {c, d}</code>	% edge optns
<code>b-- {c [&gt; "foo "]}</code>	% edge names
<code>x-&gt; (myNode) -&gt; y</code>	% ref predef'd
Options for <code>&lt;group_spec&gt;</code> ∈:	number nodes, .
Options for <code>&lt;chain_spec&gt;</code> ∈:	name, edges.
Options for <code>&lt;node_spec&gt;</code> ∈:	as, set, [source target edge style node].
Options for <code>&lt;edge_spec&gt;</code> ∈:	<color>, "<label>".

#### Macros, Colors, & Operators

From graph theory, “colors” & “operators” allow more complex / elegant graph descriptions.

<code>\graph [color class=from, ...]{</code>	% create clr
<code>[complete bipartite={x}{y}]</code>	% use in opt'r
<code>{ [x] x_1, x_2 }, { [y] y_1, y_2 } }</code>	% assign clr

Define operator with operator or default edge operator graph options. Parameter value is custom operator code like:

`declare={star}{root-- {\foreach\i in {1,...,\n} { \i} }`

... or use a pre-fab join or group operator like:

<a href="#">group</a>	<a href="#">join</a>
<a href="#">clique</a>	<a href="#">complete bipartite</a>
<a href="#">induced indep't set</a>	<a href="#">induced comp't bipartite</a>
<a href="#">cycle</a>	<a href="#">matching</a>
<a href="#">induced cycle</a>	<a href="#">matching and star</a>
<a href="#">path</a>	<a href="#">butterfly</a>
<a href="#">induced path</a>	

Macros afford common graph types:

<code>\graph { subgraph K_n [...]} % type <math>K_n</math></code>	
<code>I_n</code>	<code>K_nm</code>
<code>K_n</code>	<code>P_n</code>
<code>I_nm</code>	<code>C_n</code>
	<code>Grid_n</code>

#### Placement Strategies

Can place manually, along grid or circle, according to node sizes or node-level in graph, or with custom code. Some options for each approach:

<a href="#">manual</a>	<a href="#">grid</a>	<a href="#">size</a>	<a href="#">circular</a>
<a href="#">no placement</a>	<a href="#">chain shift</a>	<a href="#">grow left sep</a>	<a href="#">circ. placm't</a>
<a href="#">x</a>	<a href="#">gr'p shift</a>	<a href="#">grow up sep</a>	<a href="#">chain plr shift</a>
<a href="#">y</a>	<a href="#">grown up</a>	<a href="#">b'ch r't sep</a>	<a href="#">group plr shift</a>
	<a href="#">grow left</a>	<a href="#">b'ch up sep</a>	<a href="#">radius</a>
	<a href="#">b'ch up</a>		<a href="#">phase</a>
	<a href="#">b'ch left</a>		<a href="#">clockwise</a>
	<a href="#">grid placm't</a>		<a href="#">c'clockwise</a>

#### Tree

A less ornate alternative to bona-fide graphs:

<code>\node[grow'=up] {root}</code>	% various options
<code>child {node {left}}</code>	% node as child
<code>child {[fill] circle}</code>	% arb. child
<code>child[grow=100] { node}</code>	% direction ↘

Customize how parent-child edges are drawn with edge from parent node option.

## 7 Libraries

#### Angles

<code>\draw (2,0) coordinate (A) -- ...</code>	% setup
<code>pic[radius=1cm] {angle = C--B--A}</code>	% invke

#### Automata

<code>\node[state,initial](q_0)</code>	% init state
<code>\node[state](q_1)</code>	% norm'l state
<code>\node[state](q_2)[right=of q_0]</code>	% w/ opt'ns
<code>\node[state,accepting](q_4)</code>	% term state

<code>\path[-&gt;] (q_0) edge node{a}(q_1)</code>	% an edge
<code>...edge [loop above] node{b}</code>	% self-ref

Options ∈:

<a href="#">state with output</a>	<a href="#">initial distance</a>
<a href="#">state</a>	<a href="#">every initial by arrow</a>
<a href="#">every state</a>	<a href="#">initial above</a>
<a href="#">initial by arrow</a>	<a href="#">initial right</a>
<a href="#">initial text</a>	<a href="#">initial by diamond</a>
<a href="#">initial where</a>	<a href="#">accepting by double</a>

#### Background

#### Calendar

#### Entity-Relationship Diagram

#### Mind Map