Tikz
Cheatsheet, page 1 of 2

1 Tikz *(visual ref.)*

General invocations:
```
\begin{tikzpicture}[<option>*]{...
\tcbset{<option>*}
```
\begin{tikzpicture}[...]{...} % start envt
\tikz{...} % inline diagram
\tcbset{...} % Δ settings mid-course

Possible options are many, including:

- remember picture
- overlay
- [x|y] = (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-evnmt
\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
function product(\x,\y){return\x*\y;}; % func

Low-level parsing is similar to LaTeX:
\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 π
\pgfmathresult % return result

∃ *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	ifthnels	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> ∈: `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) % spher.
[canvas is xy plane at z] % draw to 2d plane

3 Path

```
\path[<option>*]
   (<from>) -- (<to>) [-{}- (<to>)]*;
```
(where <from>, <to> *are coords ... see below)*

Generally, paths start with a "move-to" operation (indicated by unpreceeded 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

∃ *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

∃ *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:*

```
\draw [-{>[sep=1pt]>}]      % separated
\draw [->_>]                % ibid
\draw [-{>>[sep=2pt]}]      % space after
\draw [<.<<->.>>]           % 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`, `open`*ed, 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

```
\node[<setting>*]
   at (<coord>)?
   (<name>?){<text>?}
```

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****:*

```
\matrix [matrix of nodes]{    % create
  1 & 2 & 3 \\                % basic syntax
  4 & 5 & |red| 6 }           % 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

```
\path[<setting>*]
   (<from>) edge[<setting>*] (<to>);}
```

... 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.

```
\graph[<graph_opt'n>*]{<graph_spec'n>}
```

```
\graph{ a -> { b, c } -> d }      % 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

```
\graph [grow down]          % placement...
\graph [branch right]       % ... 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.*

```
x1/$x_1$ -> x2 [as=$x_2$, red]  % naming
b --[thick] {c, d}              % edge optns
b -- {c [> "foo "]}             % edge names
x -> (myNode) -> y              % ref predef'd
```

Options for `<group_spec>` *∈:* `number nodes, .` *Options for* `<chain_spec>` *∈:* `name, edges.` *Options for* `<node_spec>` *∈:* `as, set, [source|target]edge[style|node].` *Options for* `<edge_spec>` *∈:* `<color>, "<label>"`.

Macros, Colors, & Operators

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

```
\graph [color class=from, ...]{     % create clr
[complete bipartite={x}{y}]         % use in optr
{ [x] x_1, x_2 }, { [y] y_1, y_2 } } % 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:

group	join
clique	complete bipartite
induced indep't set	induced comp't bipartite
cycle	matching
induced cycle	matching and star
path	butterfly
induced path	

Macros afford common graph types:

```
\graph { subgraph K_n [...]}    % type K_n
```

I_n	K_nm	Grid_n
K_n	P_n	
I_nm	C_n	

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:

manual	grid	size	circular
no placemt	chain shift	grow left sep	circ. placm't
x	gr'p shift	grow up sep	chain plr shift
y	grown up	b'ch r't sep	group plr shift
	grow left	b'ch up sep	radius
	b'ch up		phase
	b'ch left		clockwise
	grid placm't		c'clockwise

Tree

A less ornate alternative to bona-fide graphs:

```
\node[grow'=up] {root}      % various options
  child {node {left}}       % node as child
  child {[fill] circle}     % arb. child
  child[grow=100] { node}   % direction ∠
```

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

7 Libraries

Angles

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

Automata

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

\path[->] (q_0) edge node{a}(q_1)   % an edge
... edge [loop above] node{b}       % self-ref
```

Options ∈:

state with output	initial distance	
state	every initial by arrow	
every state	initial above	
initial by arrow	initial right	
initial text	initial by diamond	
initial where	accepting by double	

Background

Calendar

Entity-Relationship Diagram

Mind Map