Drawing Graphs using Dot and Graphviz

Table of Contents

tup]] : How I use Emacs Org Mode to write novels and short stories

1 License

2 Introduction

2.1 What is DOT?

DOT is a plain text graph description language. It is a simple way of describing graphs that both humans and computer programs can read.

2.2 What is Graphviz?

Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks.

2.3 Who is this document for?

This document was originally written as quick reference for myself. It was then extended to become a tutorial for Computing students. It's now offered to anyone who wants to learn DOT by example.

2.4 Related Materials

You can find links to similar documents posts at my blog TonyBallantyne.com/tech

If you have stumbled across this document by accident whilst looking for my work as an SF and Fantasy writer, the following links may be more useful to you

- TonyBallantyne.com: Blog about my novels and short stories
- Emacs Tutorial : A brief introduction to Emacs for writers
- My Emacs Writing Setup : How I use Emacs Org Mode to write novels and short stories

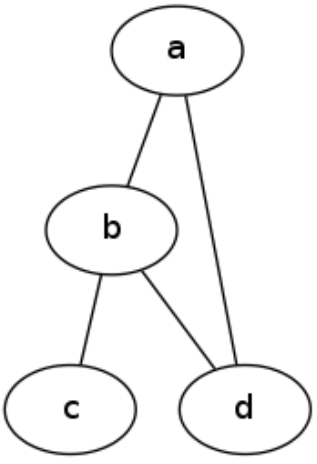
3 Setup

You will need to have the Graphviz suite of programs installed on your computer to follow this tutorial. Graphiz can be downloaded for free from the Graphviz site: http://www.graphviz.org/Home.php

4 Basic Examples

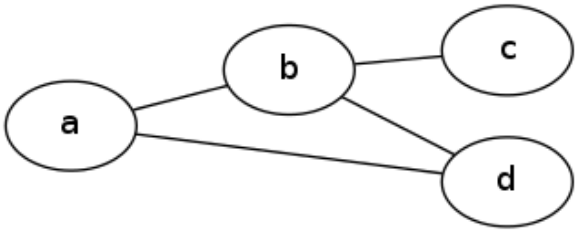
4.1 Simple Graph

```
graph graphname {
                a -- b;
                b -- c;
                b -- d;
                d -- a;
        }
```



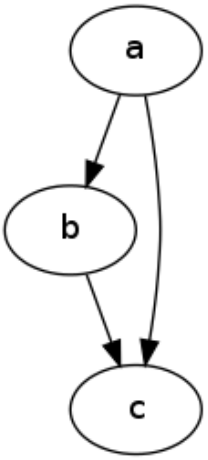
4.2 Same Graph, Different Layout

```
graph graphname {
                rankdir=LR;  //Rank Direction Left to Right
                a -- b;
                b -- c;
                b -- d;
                d -- a;
        }
```



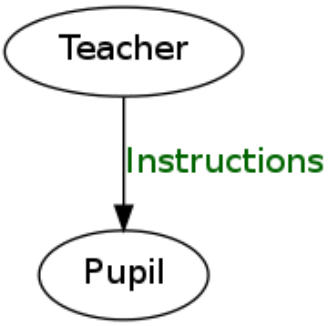
4.3 Simple Digraph (Directional Graph)

```
digraph graphname{
                a -> b;
                b -> c;
                a -> c;
        }
```



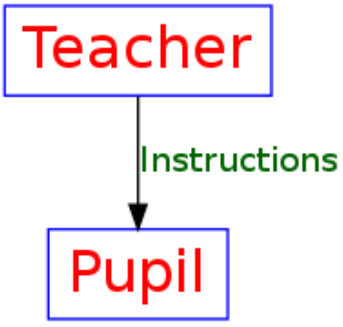
4.4 Simple Digraph with Labels

```
digraph graphname{

                T [label="Teacher"]      // node T
                P [label="Pupil"]  // node P

                T->P [label="Instructions", fontcolor=darkgreen] // edge T->P
}
```



4.5 Same Graph, Different Shape and Colour

```
digraph graphname {
                T [label="Teacher" color=Blue, fontcolor=Red, fontsize=24, shape=box]      //
                P [label="Pupil" color=Blue, fontcolor=Red, fontsize=24, shape=box]  // node P

                T->P [label="Instructions", fontcolor=darkgreen] // edge T->P
}
```

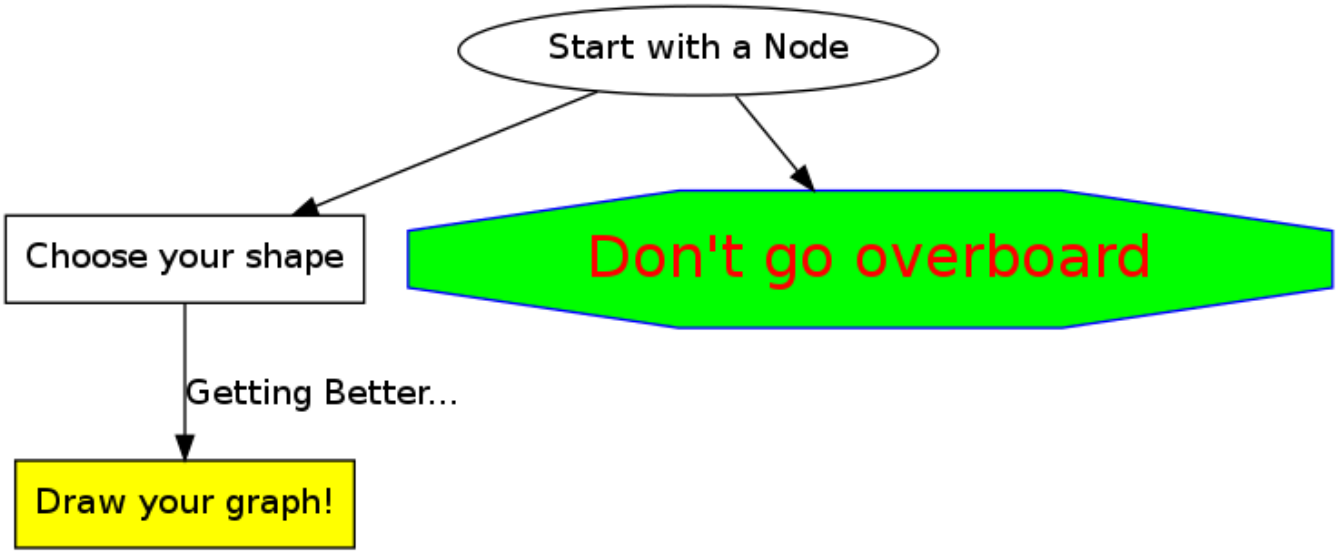


Here are some of the shapes you can use… box, polygon, ellipse, oval, circle, point, egg, triangle, plaintext, diamond, trapezium, parallelogram, house, pentagon, hexagon, septagon, octagon, doublecircle, doubleoctagon, tripleoctagon

There are lots more available here… http://www.graphviz.org/content/node-shapes

4.6 Summary

```
digraph summary{
                start [label="Start with a Node"]
                next [label="Choose your shape", shape=box]
                warning [label="Don't go overboard", color=Blue, fontcolor=Red,fontsize=24,sty
                end [label="Draw your graph!", shape=box, style=filled, fillcolor=yellow]

                start->next
                start->warning
                next->end [label="Getting Better...", fontcolor=darkblue]

}
```

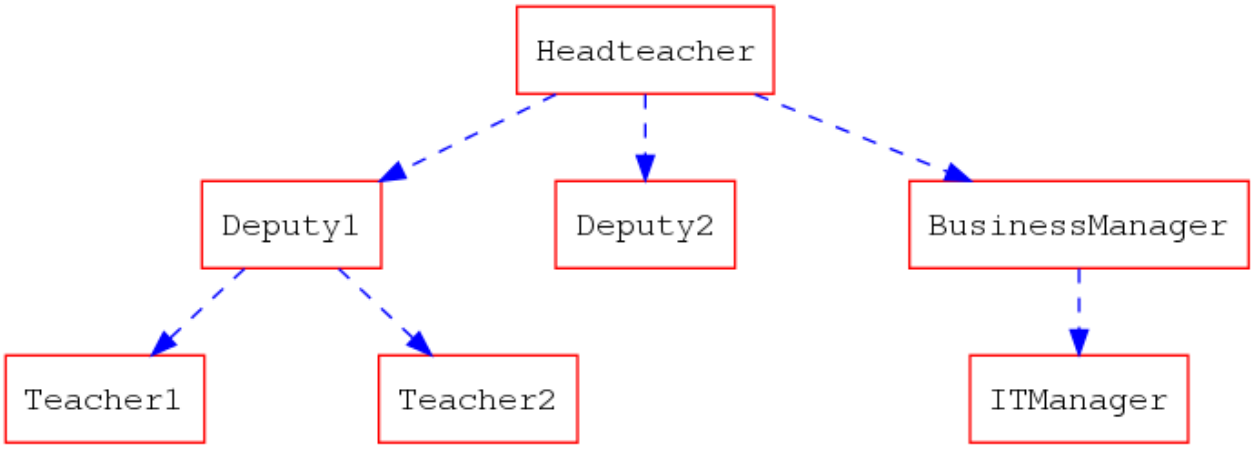


5 More Advanced

5.1 Saving Time

It takes time defining each node individually. The following way is quicker

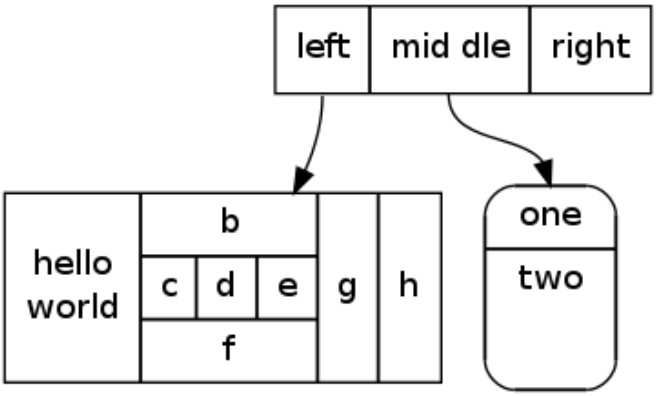
```
digraph hierarchy {

                nodesep=1.0 // increases the separation between nodes

                node [color=Red,fontname=Courier,shape=box] //All nodes will this shape and co
                edge [color=Blue, style=dashed] //All the lines look like this

                Headteacher->{Deputy1 Deputy2 BusinessManager}
                Deputy1->{Teacher1 Teacher2}
                BusinessManager->ITManager
                {rank=same;ITManager Teacher1 Teacher2}  // Put them on the same level
}
```



5.2 Records

You can now use HTML to define these sort of blocks. Find out more at http://www.graphviz.org/doc/info/shapes.html

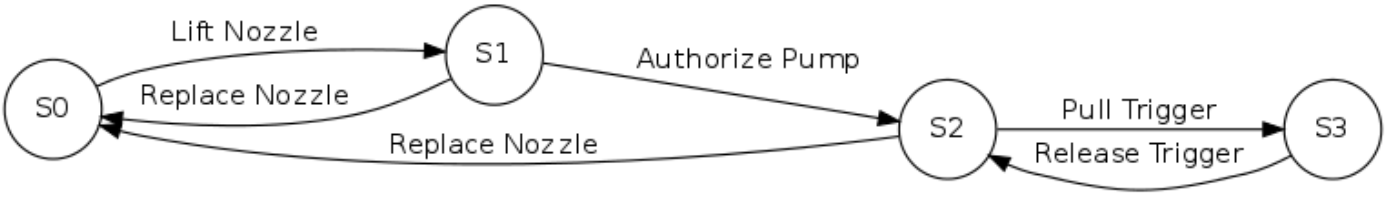
```
digraph structs {
        node[shape=record]
        struct1 [label="<f0> left|<f1> mid\ dle|<f2> right"];
        struct2 [label="{<f0> one|<f1> two\n\n\n}" shape=Mrecord];
        struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];
        struct1:f1 -> struct2:f0;
        struct1:f0 -> struct3:f1;
}
```



6 Some Example Graphs

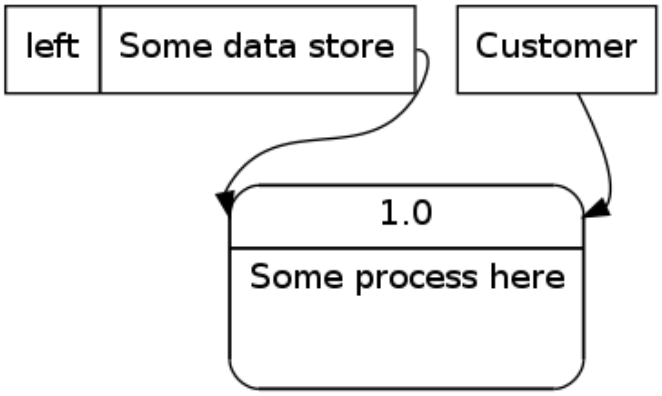
6.1 Finite State Machine

```
digraph finite_state_machine {
        rankdir=LR;
        size="8,5"
        node [shape = circle];
        S0 -> S1 [ label = "Lift Nozzle" ]
        S1 -> S0 [ label = "Replace Nozzle" ]
        S1 -> S2 [ label = "Authorize Pump" ]
        S2 -> S0 [ label = "Replace Nozzle" ]
        S2 -> S3 [ label = "Pull Trigger" ]
        S3 -> S2 [ label = "Release Trigger" ]
}
```



6.2 Data Flow Diagrams

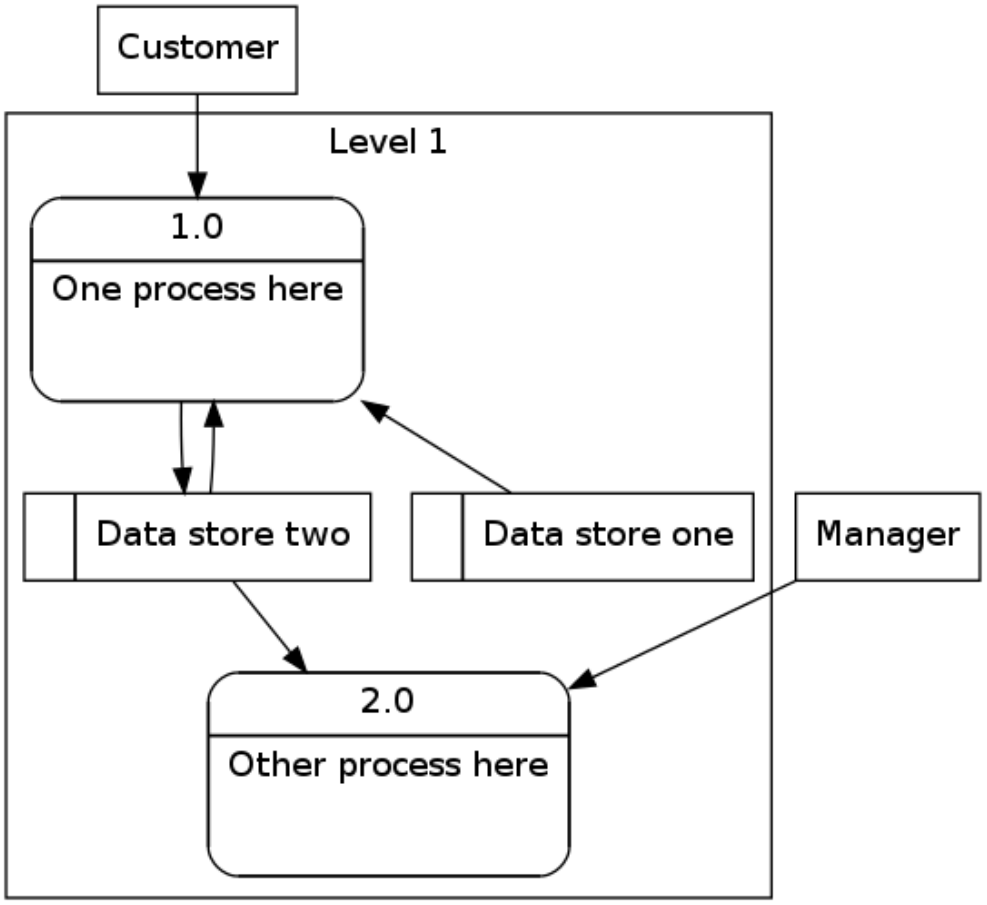
```
digraph dfd{
        node[shape=record]
        store1 [label="<f0> left|<f1> Some data store"];
        proc1 [label="{<f0> 1.0|<f1> Some process here\n\n\n}" shape=Mrecord];
        enti1 [label="Customer" shape=box];
        store1:f1 -> proc1:f0;
        enti1-> proc1:f0;
}
```



6.3 Data Flow Diagrams 2

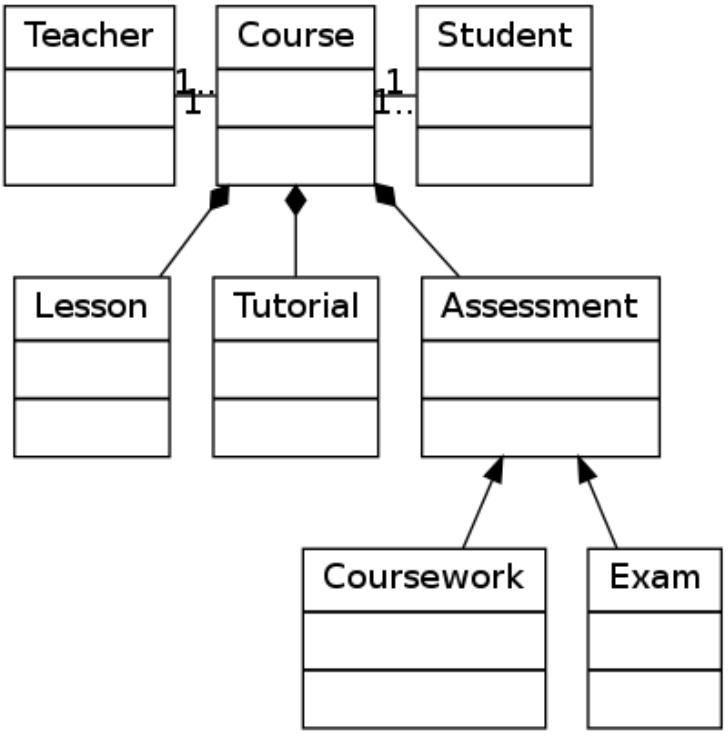
The following uses subgraphs to display different levels. Note that subgraphs must start with the prefix cluster_ or they won't work. It will only work with dot layout.

```
digraph dfd2{
        node[shape=record]
        subgraph level0{
        enti1 [label="Customer" shape=box];
        enti2 [label="Manager" shape=box];
        }
        subgraph cluster_level1{
                        label ="Level 1";
                        proc1 [label="{<f0> 1.0|<f1> One process here\n\n\n}" shape=Mrecord];
                        proc2 [label="{<f0> 2.0|<f1> Other process here\n\n\n}" shape=Mrecord]
                        store1 [label="<f0>    |<f1> Data store one"];
                        store2 [label="<f0>   |<f1> Data store two"];
                        {rank=same; store1, store2}

        }
        enti1 -> proc1
        enti2 -> proc2
        store1 -> proc1
        store2 -> proc2
        proc1 -> store2
        store2 -> proc1
}
```



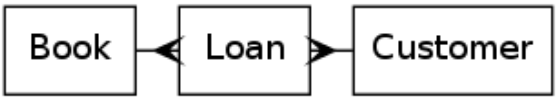
6.4 Object Inheritance

```
digraph obj{
        node[shape=record];
        rankdir="BT";

        teacher [label = "{<f0> Teacher|<f1> \n  |<f2> \n   }"];
        course [label = "{<f0> Course|<f1> \n  |<f2> \n   }"];
        student [label = "{<f0> Student|<f1> \n  |<f2> \n   }"];
        lesson [label = "{<f0> Lesson |<f1> \n  |<f2> \n   }"];
        tutorial [label = "{<f0> Tutorial|<f1> \n  |<f2> \n   }"];
        assessment[label = "{<f0> Assessment|<f1> \n  |<f2> \n   }"];
        coursework [label = "{<f0> Coursework|<f1> \n  |<f2> \n   }"];
        exam [label = "{<f0> Exam|<f1> \n  |<f2> \n   }"];

        {rank=same; teacher course student}

        teacher->course [dir="forward",arrowhead="none",arrowtail="normal",headlabel="1",taill
        student->course [dir="forward",arrowhead="none",arrowtail="normal",headlabel="1",taill
        lesson->course [dir="forward",arrowhead="diamond",arrowtail="normal"];
        tutorial->course [dir="forward",arrowhead="diamond",arrowtail="normal"];
        assessment->course [dir="forward",arrowhead="diamond",arrowtail="normal"];
        coursework->assessment;
        exam->assessment;

}
```



6.5 Entity Relationship

```
digraph ER{
        node[shape=box];
        Book;
        Customer;
        Loan;
        {rank=same;Book,Customer,Loan}
        Book->Loan[dir="forward",arrowhead="crow",arrowtail="normal"];
        Customer->Loan[dir="forward",arrowhead="crow",arrowtail="normal"];
}
```



7 Reference

Here are the most useful attributes you will need when drawing graphs. The full list can be found here: http://graphviz.org/doc/info/attrs.html

7.1 Graph Attributes

- label="My Graph"; Label a graph itself
- rankdir=LR; Lay the graph out from Left to Right, instead of Top to Bottom
- {rank=same; a, b, c } Group nodes together at the same level of a graph
- splines="line"; Force edges to be straight, no curves or angles
- K=0.6; Used to influence the 'spring' used in the layout, Can be used to push nodes further apart, which is especially useful for twopi and sfdp layouts

7.2 Vertex Attributes

- [label="Some Label"] Labels the Vertex
- [color="red"] Colors the Vertex
- [fillcolor="blue"] Fills the Vertex with the specified colour

7.3 Edge Attributes

- [label="Some Label"] Labels the Edge (Useful for Weights)
- [color="red"] Colors the Vertex (Useful for Paths)
- [penwidth=2.0] Adjusts the thickness of the edge line, Very useful for Paths

7.4 Size, Background Colour

fixedsize=true; size="1,1"; resolution=72; bgcolor="#C6CFD532";

8 Appendices

8.1 Further Reading

- http://linuxdevcenter.com/pub/a/linux/2004/05/06/graphviz_dot.html
- http://graphs.grevian.org/index.html

8.2 Using Emacs Org Mode

Emacs org mode is an ideal environment for writing, executing and exporting Dot graphics

8.2.1 Setup

Download and install graphviz and add the path to the exec-path variable

You will need to update your .emacs file to load dot as a babel language. The following is a useful babel setup for dot and other languages

```
(org-babel-do-load-languages
 (quote org-babel-load-languages)
 (quote ((emacs-lisp . t)
         (java . t)
         (dot . t)
         (ditaa . t)
         (R . t)
         (python . t)
         (ruby . t)
         (gnuplot . t)
         (clojure . t)
         (sh . t)
         (ledger . t)
         (org . t)
         (plantuml . t)
         (latex . t))))
```

8.2.2 Embedding Dot in Emacs

Org mode can interpret different languages by using the Library Of Babel. To do so, enclose the code in begin_ src and end_ src tags as below. You'll need to add command line arguments as shown.

A shortcut to make a begin_ src block is to type <s [TAB]

```
#+begin_src dot :file ./img/example1.png :cmdline -Kdot -Tpng
graph graphname {
                a -- b;
                b -- c;
                b -- d;
                d -- a;
```

```
        }
#+end_src
```

8.2.3 The Command Line

The section :cmdline -Kdot -Tpng in the #+begin_ src dot :file ./img/example1.png :cmdline -Kdot -Tpng section are command line arguments. They tell dot how to render and display.

- -Kdot use dot layout. The other layouts you can try are Kneato, Kcirco, Ktwopi, Kfdp and Ksfdp for different layouts
- -Tpng render as png

The full command line arguments can be found here: http://graphviz.org/content/command-line-invocation

Date: <2013-10-21 Mon>
Author: Tony Ballantyne
Created: 2014-04-12 Sat 10:13
Emacs 23.3.1 (Org mode 8.0.2)
Validate XHTML 1.0