An excerpt from

# The Graphviz Cookbook

Rod Waldhoff
rwaldhoff@gmail.com
http://heyrod.com/

**ABOUT THIS BOOK**

*The Graphviz Cookbook*, like a regular cookbook, is meant to be a practical guide that *shows you how to create something tangible* and, hopefully, *teaches you how to improvise your own creations* using similar techniques.

The book is organized into four parts:

***Part 1: Getting Started*** introduces the Graphviz tool suite and provides "quick start" instructions to help you get up-and-running with Graphviz for the first time.

***Part 2: Ingredients*** describes the elements of the Graphviz ecosystem in more detail, including an in-depth review of each application in the Graphviz family.

***Part 3: Techniques*** reviews several idioms or "patterns" that crop up often when working with Graphviz such as how to tweak a graph's layout or add a "legend" to a graph. You might think of these as "micro-recipes" that are used again and again.

***Part 4: Recipes*** contains detailed walk-throughs of how to accomplish specific tasks with Graphviz, such as how to spider a web-site to generate a sitemap or how to generate UML diagrams from source files.

Distribution Code 15343

Chapter 1

# Graphviz in a Nutshell

Graphviz is a suite of *graph vis*ualization and manipulation tools. Using Graphviz you can convert a text-based description of a graph, like this:

```
graph G {
  A -- B
  A -- C
}
```

**Figure 1.1:** *A simple Graphviz file.*

into an image, like this:

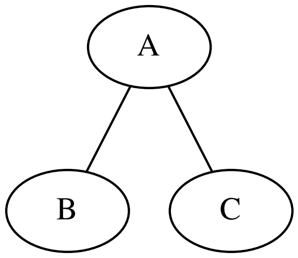

**Figure 1.2:** *An image representing the graph described in Figure 1.1.*

But that's just one of many possible ways to render that graph. Graphviz, and its associated tool-chain, can create a staggeringly diverse collection of renditions of that simple graph.

Graphviz is open source software, first developed by AT&T Labs Research. Graphviz is available for Linux (and other Unix-like), Microsoft Windows and

Apple's OS X operating systems. You can obtain, use and even redistribute Graphviz for free, whether for personal or commercial purposes.[1]

Graphviz's home on the web is http://graphviz.org.

## 1.1 Components

Graphviz primarily consists of:

- **The DOT graph description language** – a way to describe graphs as plain text documents. Figure 1.1 is simple example of a DOT document. By convention, DOT files have a `.gv` extension, but `.dot` was once more commonly used.[2]
- **Several layout engines** – programs that convert a DOT document into an image, or more broadly, into a "layout" that specifies where to place each node and edge. These include `dot`, `neato`, `twopi`, `circo`, `fdp`, `sfdp` and `patchwork`.
- **Post-processing and pre-processing tools** – programs that that manipulate and filter DOT files (including `gvcolor`, acyclic, `unflatten`, `gvpack`, `prune` and others).
- **Graph-generation and analysis tools** – programs that generate graphs interactively (e.g., `dotty`) or from scratch (e.g., `gvgen`) or that analyze graphs to extract certain pieces of information (including `gc`, `gvpr`, `ccomps`, `dijkstra` and others).

In addition, Graphviz is at the center of a large ecosystem of third-party tools that can generate, manipulate, render and embed DOT files and Graphviz images.

We'll cover many core and third party components in more depth as we go along.

## 1.2 Quick Start

Let's get our hands dirty and create some graphs.

### Your First Graph

Fire up your favorite text editor and create a file like `mygraph.gv` shown in Figure 1.3.

[1] See "A word on licensing " (Appendix C) for more on Graphviz licensing.
[2] The `.dot` extension is now more commonly associated with Microsoft Word "template" files.

```
digraph G {
  A -> B -> C
  B -> D
}
```

**Figure 1.3:** *Your first DOT file. Save this into a text file named* `mygraph.gv`*.*

Now open up your terminal emulator and run:[3]

```
> dot -Tpng mygraph.gv -o mygraph.png
```

**Figure 1.4:** *Invoking* `dot` *to create an image from your DOT file.*

This will generate a PNG-format image file named `mygraph.png` containing a representation of your graph. Open it with your web browser, image-viewer or graphics program.[4] The image I get is shown in Figure 1.5.

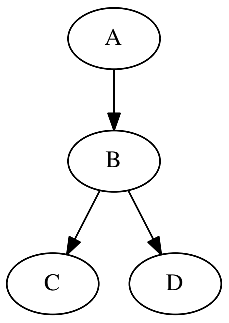

**Figure 1.5:** *Your first DOT file (*`mygraph.gv`*), rendered with the* `dot` *layout engine via the command in Figure 1.4.*

## More Nodes and Edges

Let's add a few more nodes and edges to our graph, just to see how Graphviz changes the layout. Modify `mygraph.gv` as shown in Figure 1.6.

[3] This section assumes you already have Graphviz installed. If you need help with that, see Chapter 2.

[4] If you're on Linux or another Unix derivative, an even faster way to quickly view the graph generated by `dot` or other layout engine is to use the `-Txlib` or `-Tgtk` output option. These will immediately launch an X11{} or GTK window (respectively) containing a representation of the graph. You don't even need to write to an output file (`-o`). See "Output Formats " (Chapter 5) for more information.

```
digraph G {
  A -> B -> C
  B -> D

  D -> E -> G
  D -> F -> G

  G -> H
  G -> I

  go -> yield -> stop

  C -> yield
}
```

**Figure 1.6:** *Adding a few more nodes and edges to the graph. The added content is emphasized.*

Now when we run the `dot` command in Figure 1.4 we get the image in Figure 1.7.

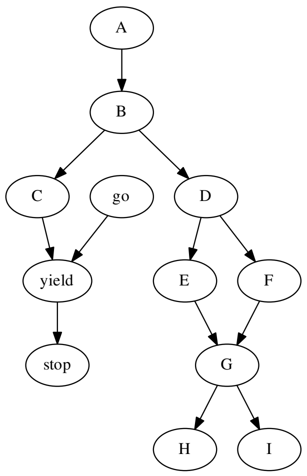

**Figure 1.7:** *The graph with a few more nodes and edges (Figure 1.6), rendered with the* `dot` *layout engine.*

## Adding Flair

Now let's really mix it up by adding various color and style attributes to the graph, as seen in Figure 1.8.

```
digraph G {
  B [fontsize=26]

  A -> B -> C
  B -> D

  node [style=filled]
  E [shape=box, fillcolor="#dddddd"]
  F [shape=folder, fillcolor=goldenrod]
  G [shape=none, style=""]
  H [shape=star, fillcolor=darkseagreen,
   style="filled,rounded"]
  I [shape=star, color=dodgerblue]

  D -> E -> G
  D -> F -> G

  G -> H
  G -> I

  subgraph clusterTraffic {
    margin=16
    node [style="filled,bold" height=0.8, width=0.8,
          color=black, fontname=Sans, fixedsize=true]
    go [label="GO", shape=circle, fillcolor=green]
    yield [label="YIELD", shape=triangle, height=1.2,
           width=1.2, style="filled,bold,rounded",
           fillcolor=yellow]
    stop [label="STOP", shape=polygon, sides=8,
          color=black, fillcolor=red]
  }

  go -> yield -> stop

  edge [style=dotted, arrowhead=odot]

  C -> yield
}
```

**Figure 1.8:** *Adding flair to our graph using node, edge and graph attributes. Note that none of the original lines in the file have changed. The added content is emphasized.*

Now when we run the `dot` command in Figure 1.4 we get an image like that in Figure 1.9.

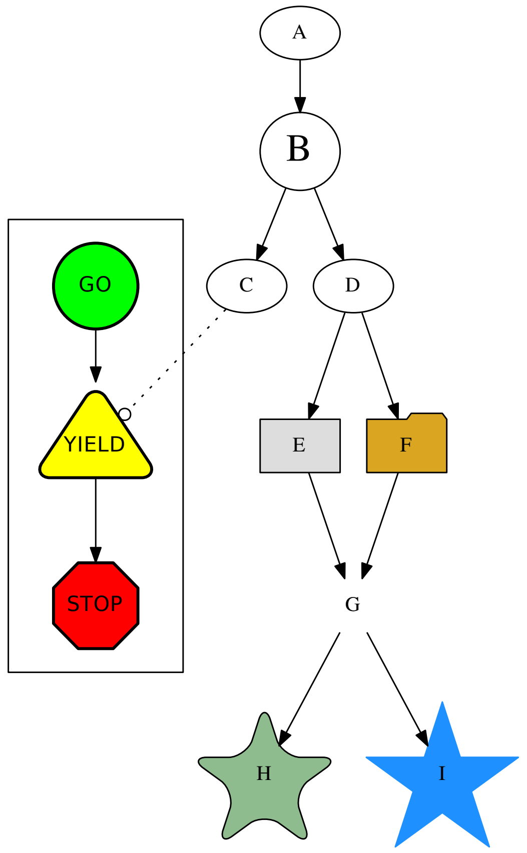

**Figure 1.9:** *Our graph with added flair, rendered with the* `dot` *layout engine.*

Don't worry if some (or all) of the code in Figure 1.8 doesn't make sense to you right now. Chapter 4 covers all this and more in great detail.

*Do* note that Figure 1.6 and Figure 1.8 contain the exact same nodes and edges. We've added styling and grouping attributes, but all of the lines in Figure 1.6 appear unchanged in Figure 1.8.

## Alternative Layouts

Just for fun, let's see how some of the other layout engines render our graph. The command:

```
> neato -Tpng mygraph.gv -o mygraph.png
```

**Figure 1.10:** *Invoking the layout engine* `neato` *to create an image from the DOT file.*

uses the `neato` layout engine instead of `dot`, generating Figure 1.11.

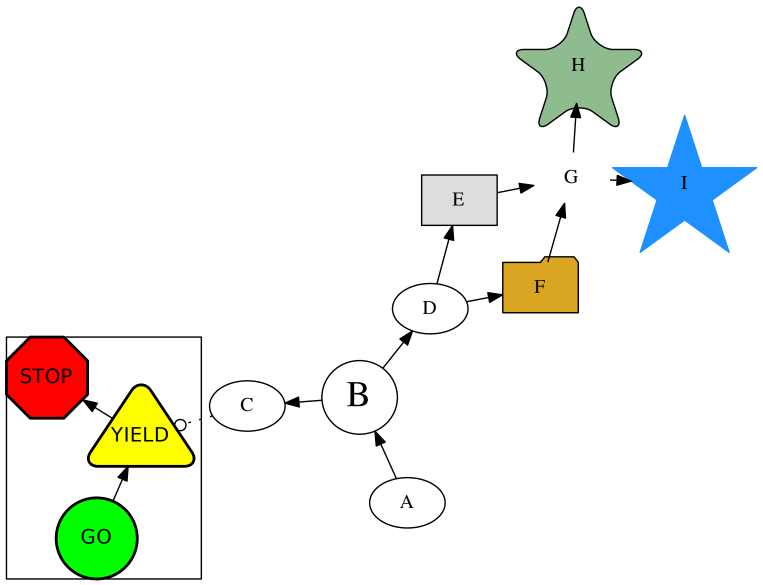

**Figure 1.11:** `mygraph.gv` *rendered with the* `neato` *layout engine.*

The command:

```
> circo -Tpng mygraph.gv -o mygraph.png
```

**Figure 1.12:** *Invoking the layout engine* `circo` *to create an image from the DOT file.*

uses the `circo` layout engine, generating Figure 1.13.

The command:

```
> patchwork -Tpng mygraph.gv -o mygraph.png
```

**Figure 1.14:** *Invoking the layout engine* `patchwork` *to create an image from the DOT file.*

uses the `patchwork` layout engine, generating the wild image found in Figure 1.15.

## 1.3 What next?

- Visit "The DOT Language in Depth" (Chapter 4) for an in-depth look at the DOT language and what you can do with it.

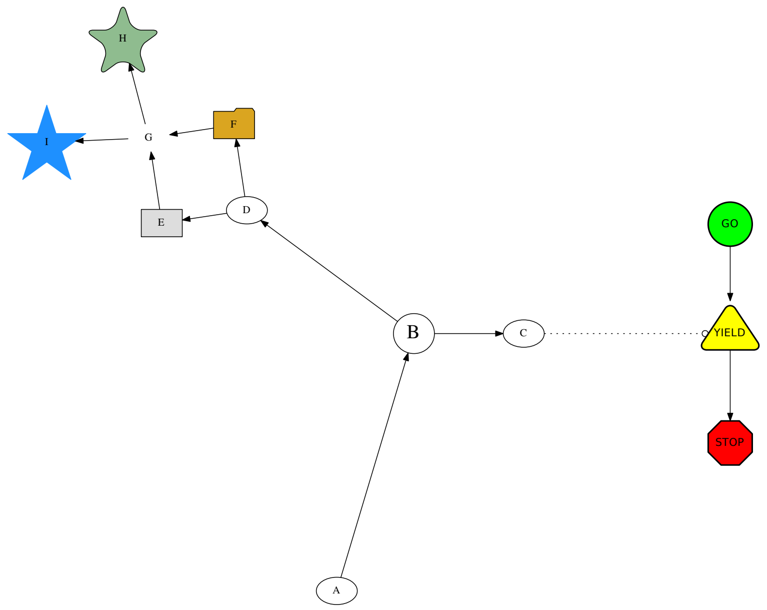

**Figure 1.13:** `mygraph.gv` *rendered with the* `circo` *layout engine.*

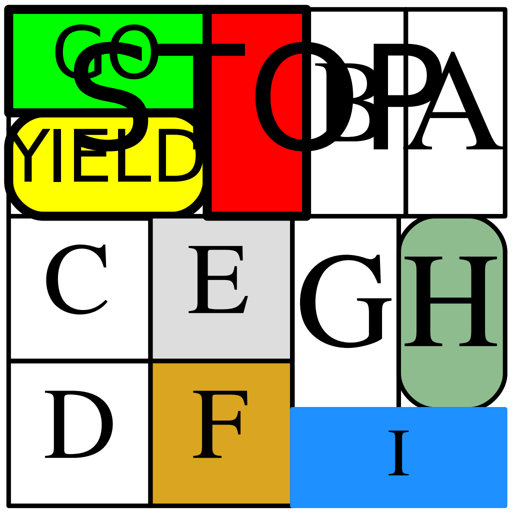

**Figure 1.15:** `mygraph.gv` *rendered with the* `patchwork` *layout engine.* `patchwork` *performs a different kind of layout than the other engines we've tried, and as you can see, our graph isn't quite set up to be rendered by* `patchwork` *very well just yet. While this might be an interesting little piece of abstract art, we'll discuss how to use* `patchwork` *properly in Section 6.7.*

- Visit "Layout Engines " (Chapter 6) to learn more about the different layout engines that make up the Graphviz family.
- Visit "The Rest of the Graphviz Suite" (Chapter 7) to learn more about the other programs and utilities that are bundled with Graphviz.
- If you're feeling adventurous, don't be afraid to experiment a bit on your own. Make some changes to the graph you created in `mygraph.gv` file to see what impact they have on the rendered image. Experiment with the different layout engines and their command line options. Graphviz has an extensive set of `man` pages. Try `man graphviz` and `man dot` to start.
- If you feel you've got a pretty good handle on Graphviz and the DOT language, skip ahead to "Techniques" (Part III) and "Recipes" (Part IV) and see if anything catches your eye.
- Read on for more background information about the world of Graphviz.