Inhaltsverzeichnis

Vereinfachte Abhängigkeiten mit Graphviz

Graphviz ist ein ziemlich geniales Tool, um Zusammenhänge (neudeutsch: Graphen) auf's Papier zu bringen. Aber so mächtig, wie das Tool ist, so kompliziert wird die Syntax und vor allem das Linien-Ziehen zwischen den Nodes, wo man mächtig aufpassen muß, sich in den Node- IDs nicht zu verheddern.

Ausserdem wird das Ganze auch ein ziemlicher Knoten, wenn ein paar zentrale Notes die Wurzel für viele Abhängigkeiten sind und damit so ziemlich überall hin verbunden werden müssen.

Und wenn man schon mal dabei ist, wäre es auch schön, wenn man nach Bedarf mehrere Dateien laden könnte und sie zu einer Übersicht zusammen zu fügen, um verschiedene Ansichten erzeugen zu können, ohne die Details parallel in mehreren Graphviz- Quelldateien pflegen zu müssen.

Daraus entstand dann

Simple_Graphviz

Simple_Graphviz ist ein kleines Python- Script, was folgendes macht:

- es liest eine oder mehrere Eingabedateien ein, die in einer einfachen Syntax beschreiben, welche Elemente von welchen anderen Elementen abhängen
- Wenn vor den Namen eines Elements ein führendes _ (Underline) gesetzt wird, wird dieses Element als Signal betrachtet.

Das Ergebnis wird in die Standard-Ausgabe geschrieben, als kleine Erleichterung kopiert das Programm das Ergebnis auch direkt ins Clipboard zur Weiterverwendung, wenn es im Betriebssystem die nötigen Clipboard- Treiber finden kann.

Alle Signal- Elemente werden als Pfeile gezeichnet werden und nicht mehr mit Ihrem Wurzel- Element verbunden, statt dessen tauchen sie überall, wo sie referenziert werden, als eigenständiger Knoten auf, so dass das die Diagramme durch weniger Striche und Ebenen übersichtlicher werden.

Die Eingabe- Syntax

Vom Format her sind die Eingabe- Dateien in YAML formatiert:

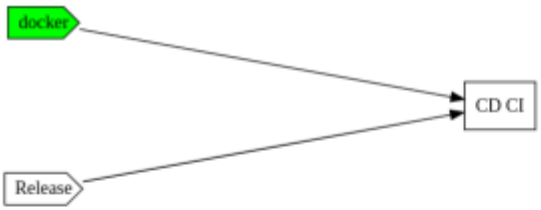
simple_graphviz.yaml

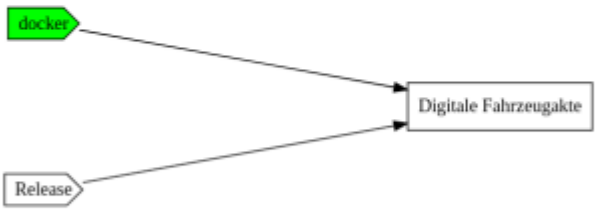
```
"_docker":
  c: green
  deps:
    - Server
    - gitlab
"Digitale Fahrzeugakte":
  deps:
```

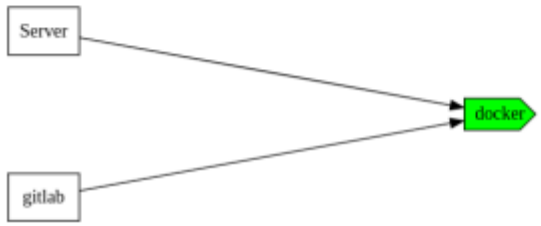
```
    - docker
    - Release
"CD CI":
  deps:
    - docker
    - _Release
```

Wie man erahnen kann, hat jedes Element seine `deps`, das sind die Elemente, von denen das jeweilige Element abhängig ist. Es gibt auch noch die optionalen Parameter `c` und `s`, mit denen man die Farbe und den Shape des Knotens in der Darstellung setzen kann. Signale werden aber immer als Pfeil gezeichnet.

Wie man auch sehen kann, sind zwei Elemente (`_docker` und `_Release`) per Unterstrich _ als Signale gekennzeichnet.







Das Programm

Nichts besonderes: Das Python- Programm benötigt die Pakete für YAML und fürs Clipboard (am besten in einer virtuellen .venv- Umgebung)

```
pip install clipboard pyyaml
```

Ohne Angabe von Eingabedateien wird defaultmäßig nach simple_graphviz.yaml gesucht, ansonsten werden die angegebenen Eingabedateien zu einer gemeinsamen Ausgabe zusammengefasst.

```
python simple_graphviz.py [Eingabe_1.yaml ...]
```

und weil's sich nicht lohnt, dafür ein eigenes Repository aufzumachen, hier das Listing

simple_graphviz.py

```
import sys
import os
import yaml
import clipboard

output = ""
index = 1
nodes = {}


def collect_output(text):
    global output
    output += str(text)+os.linesep


def print_output():
    global output
    print(output)
    try:
        clipboard.copy(output)
        print("result copied into clipboard")
    except:
        pass


def init_node(name, is_signal):
    global index, nodesn
    if name not in nodes:
        nodes[name] = {
            "index": index,
            "style": "box",
            "color": "white",
            "deps": [],
            "label": name,
            "signal": is_signal
        }
        index += 1
        return True
    if is_signal: # when a dependency node is been written as signel
```

```
anywhere, the note is set as signal globally
        nodes[name]["signal"]=True
    return False


def transform_nodes(nodes_raw):
    global nodes

    for node_raw_name, node_raw_data in nodes_raw.items():
        if node_raw_name[:1] == "_":
            node_raw_name = node_raw_name[1:]
            is_signal = True
        else:
            is_signal = False
        # if the node was defined before through a dependency, it might
miss it's signal flag
        if not init_node(node_raw_name, is_signal) and is_signal:
            nodes[node_raw_name]["signal"] = is_signal
        this_node = nodes[node_raw_name]
        for node_key, node_value in node_raw_data.items():
            if node_key == "s":
                this_node["style"] = node_value
            elif node_key == "c":
                this_node["color"] = node_value
            if node_key == "deps":
                for dep_node_name in node_value:
                    if dep_node_name[:1] == "_":
                        real_node_name = dep_node_name[1:]
                        node_is_signal = True
                    else:
                        node_is_signal = False
                        real_node_name = dep_node_name
                    init_node(real_node_name, node_is_signal)
                    this_node["deps"].append(dep_node_name)


def create_output():
    global nodes,index
    # first we do the root ones
    collect_output("digraph G {")
    collect_output('''ratio="fill";
 size="8.3,11.7!";
 rankdir="LR";
 margin=0;''')
    for node_name, node_data in nodes.items():
        if node_data["signal"]:
            collect_output("{} [shape=cds, style=filled, fillcolor={},
label=\"{}\"]".format(
                "L"+str(node_data["index"]), node_data["color"],
node_name))
        else:
```

```
            collect_output("{} [shape={},  style=filled, fillcolor={},
label=\"{}\"]".format(
                "L"+str(node_data["index"]), node_data["style"],
node_data["color"], node_name))
        # now we do the connections and create the signal nodes
    for node_data in nodes.values():
        for dep_node_name in node_data["deps"]:
            if dep_node_name[:1] == "_":
                dep_node_name = dep_node_name[1:]
            signal_color=nodes[dep_node_name]["color"]
            if nodes[dep_node_name]["signal"]:
                collect_output("{} [shape=cds, style=filled,
fillcolor={}, label=\"{}\"]".format(
                "L"+str(index), signal_color, dep_node_name))
                collect_output("{} -> {}".format(
                "L"+str(index), "L"+str(node_data["index"])))
                index+=1
            else:
                dep_node_index=nodes[dep_node_name]["index"]
                collect_output("{} -> {}".format(
                "L"+str(dep_node_index),  "L"+str(node_data["index"])))


    collect_output("}")


if len(sys.argv) > 1:
    input_file_path_array = sys.argv[1:]  # all args except [0] (script
path)
else:
    input_file_path_array = ["simple_graphviz.yaml"]
for input_file_path in input_file_path_array:
    with open(input_file_path, "r") as stream:
        try:
            nodes_raw = yaml.safe_load(stream)
            transform_nodes(nodes_raw)

        except yaml.YAMLError as exc:
            print(exc)
create_output()
print_output()
```