Resolution Test

ANGULAR SCHEMATICS

THE UNKNOWN HERO

2015, Tracey Williams Photography
llow me on Twitter @upaday

CHRIS NORING

CLOUD DEVELOPER EXPERT AT MICROSOFT

Google Developer Expert in Angular and Web Techs
@chris_noring
https://softchris.github.io
Published author
Public Speaker
Software Engineer

WORKFLOW TOOL

CAN APPLY TRANSFORMS TO YOUR PROJECT

example of usage:

- scaffold: new components, services,
- update your code to fix breaking changes in dependencies.
- or add a new configuration option or framework to an existing project

ANGULAR CLI - SECRET SAUCE

GENERIC FACILITY TO SUPPORT THE CLI SCAFFOLDING

improve your development productivity

- Ease uf use and development
- Extensibility and reuseability
- Atomicity
- Asynchronicity

TREE

OVER FILE SYSTEM

not affecting the file system directly, you describe what transformations you want done on a Tree

INSTALLATION

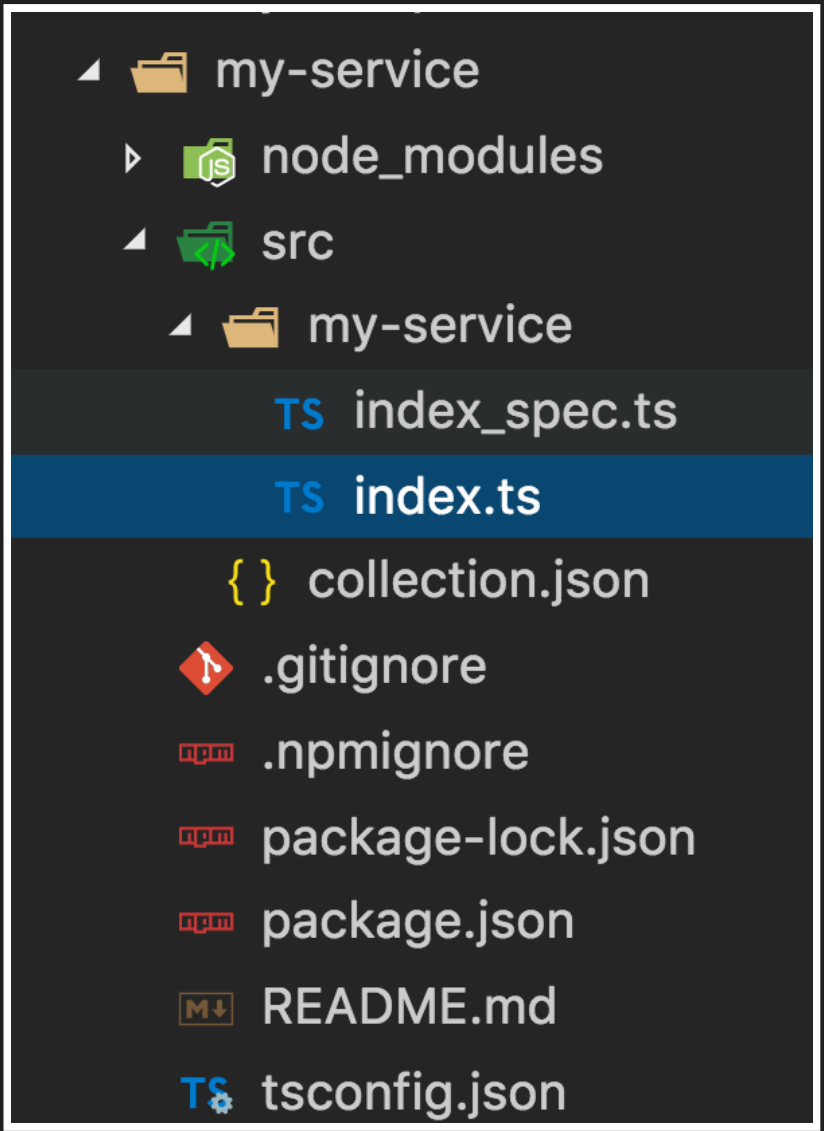
```
npm install -g @angular-devkit/schematics-cli
```

This will install a `schematics` executable, which you can use to create a blank Schematics project

EXAMPLE

YOUR FIRST SCHEMATIC

```
schematics blank --name=my-component
```


my-service
node_modules
src
my-service
index_spec.ts
index.ts
collection.json
.gitignore
.npmignore
package-lock.json
package.json
README.md
tsconfig.json

- index.ts, where our rules live
- collection.json, a description of our project

INDEX.TS

- return a function of type **Rule**
- **options** input, is an object representing command line args
- inner function takes **Tree** and **SchematicContext**, tree is file tree

index.ts

```
1   import { Rule, SchematicContext, Tree } from '@angular-devkit/schematics';
2
3
4   // You don't have to export the function as default.
5   // You can also have more than one rule factory
6   // per file.
7   export function myService(_options: any): Rule {
8     return (tree: Tree, _context: SchematicContext) => {
9       return tree;
10    };
11  }
12
```

COLLECTION.JSON

- **description**, description of project
- **factory**, points to the directory/starter module

collection.json

```
1  {
2    "$schema": "../node_modules/@angular-devkit/schematics/collection-schema.json",
3    "schematics": {
4      "my-service": {
5        "description": "A blank schematic.",
6        "factory": "./my-service/index#myService"
7      }
8    }
9  }
10
```

BUILD AND RUN

```
npm run build
schematics .:my-component --name=test
```

--name=test, command line arg

```
➜ my-service npm run build

> my-service@0.0.0 build /Users/chnoring/Documents/dev/projects/schematics-project/my-service
> tsc -p tsconfig.json

➜ my-service schematics .:my-service --name=test
Nothing to be done.
➜ my-service
```

it says: nothing to be done!!

not surprised, cause our Rule factory function does nothing

EXAMPLE

SCAFFOLD SOME FILES, WITH CONTENT

update Rule factory function

create files, with content

```
import { Rule, SchematicContext, Tree } from "@angular-devkit/schematics";

// You don't have to export the function as default.
// You can also have more than one rule factory
// per file.
export function myService(_options: any): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    tree.create("maxim.ts", "is a viking");
    tree.create("sherry.ts", "is a viking");
    tree.create("ana.ts", "is a viking");
    tree.create("chris.ts", "is a viking");
    return tree;
  };
}
```

tree.create(filename, file-content)

let's create a build and run script

```
"scripts": {
  "build": "tsc -p tsconfig.json",
  "test": "npm run build && jasmine src/**/*_spec.js",
  "execute": "npm run build && schematics .:my-component --name=test"
},
```

```
npm run execute
```

```
➜  my-service npm run execute

> my-service@0.0.0 execute /Users/chnoring/Documents/dev/projects/schematics-project/my-service
> npm run build && schematics .:my-service --name=test


> my-service@0.0.0 build /Users/chnoring/Documents/dev/projects/schematics-project/my-service
> tsc -p tsconfig.json

CREATE /maxim.ts (11 bytes)
CREATE /sherry.ts (11 bytes)
CREATE /ana.ts (11 bytes)
CREATE /chris.ts (11 bytes)
➜  my-service
```

we didn't actual create any files, we are in dry-run mode

It shows what would happen if applied to a file system

let's shut off dry-run

```
  "execute": "npm run build && schematics .:my-service --name=test",
  "execute:prod": "npm run build && schematics .:my-service --name=test --dry-run=false"
},
```

I don't see a difference??

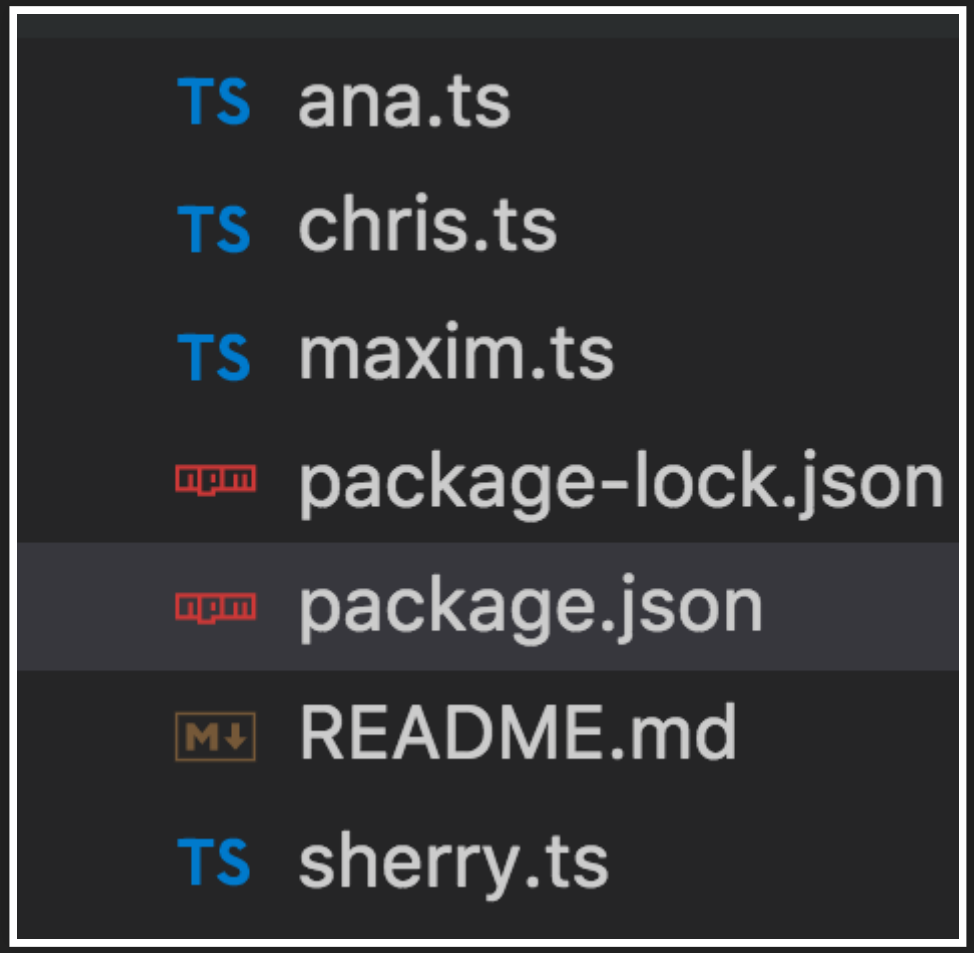
```
➜  my-service npm run execute:prod

> my-service@0.0.0 execute:prod /Users/chnoring/Documents/dev/projects/schematics-project/my-service
> npm run build && schematics .:my-service --name=test --dry-run=false


> my-service@0.0.0 build /Users/chnoring/Documents/dev/projects/schematics-project/my-service
> tsc -p tsconfig.json

CREATE /maxim.ts (11 bytes)
CREATE /sherry.ts (11 bytes)
CREATE /ana.ts (11 bytes)
CREATE /chris.ts (11 bytes)
➜  my-service
```

oh, so they were created in the in the project



Just add the follwing to your `schematics` executable

```
--dry-run=false
```

There are four methods that directly create a change in a Tree

- create
- delete
- rename
- overwrite

TIPS AND TRICKS

watch mode, rebuild your project on changes

```
npm run build -w
```

Debugging

```
node --inspect-brk <path_to_schematics>/schematics
.:myComponent --name=test
```

TESTING OUT YOUR SCHEMATICS

- scaffold an Angular project
- npm link to the schematics
- ng generate

```
ng new SchematicsDemo
```

```
npm link <path to schematics>
// tip: run pwd and copy result
```

```
ng generate my-service:my-service someName
```

.gitignore
ana.ts
angular.json
chris.ts
maxim.ts
package-lock.js
package.json
README.md
sherry.ts

```
1    is a viking
```

Voila, our Vikings are added

EXAMPLE

USING AN EXISTING SCHEMATIC

WHAT IS A COLLECTION?

- Schematics Collections are sets of named schematics, that are published and installed by users.
- Angular team publishes and maintains the official @schematics/angular collection e.g component, module, application

Scenario: add some disclaimer text on top of component

We start by scaffolding a project

```
schematics blank --name=my-external
```

We need to add two new helper functions:

- **chain**, is a RuleFactory provided by Schematics library, chains multiple rules together
- **externalSchematic**, also a rule factory

```
import {
  Rule,
  SchematicContext,
  Tree,
  chain,
  externalSchematic } from '@angular-devkit/schematics';
```

install @schematics/angular

```
externalSchematic('@schematics/angular', 'component', options)
```

imports and runs the schematic

Next part is our schematic that needs to

- go through all files
- for each file list it's content and add header text

```
const headerText = `
@copyright ngVikings, by the community, for the community
`
```

```
(tree: Tree, _context: SchematicContext) => {
  tree.getDir(options.sourceDir)
    .visit(filePath => {
      if (!filePath.endsWith('.ts')) {
        return;
      }
      const content = tree.read(filePath);
      if (!content) {
        return;
      }

      // Prevent from writing license to files that already ha
      if (content.indexOf(licenseText) == -1) {
        tree.overwrite(filePath, headerText + content);
      }
```

Building and running the above

```
➜  schematics-demo git:(master) ✗ ng generate my-external:my-external madViking
? What name would you like to use for the component? madViking
CREATE src/app/mad-viking/mad-viking.component.css (0 bytes)
CREATE src/app/mad-viking/mad-viking.component.html (29 bytes)
CREATE src/app/mad-viking/mad-viking.component.spec.ts (650 bytes)
CREATE src/app/mad-viking/mad-viking.component.ts (284 bytes)
UPDATE src/app/app.module.ts (1013 bytes)
```

A scaffolded component

with our header text in place

```
/**
 * @copyright
 * ngVikings, by the community, for the community
 */
import { Component, OnInit } from "@angular/core";

@Component({
  selector: "app-mad-viking",
  templateUrl: "./mad-viking.component.html",
  styleUrls: ["./mad-viking.component.css"]
})
export class MadVikingComponent implements OnInit {
  constructor() {}

  ngOnInit() {}
}
```

EXAMPLE

SCAFFOLD BASED ON TEMPLATES

We want to:

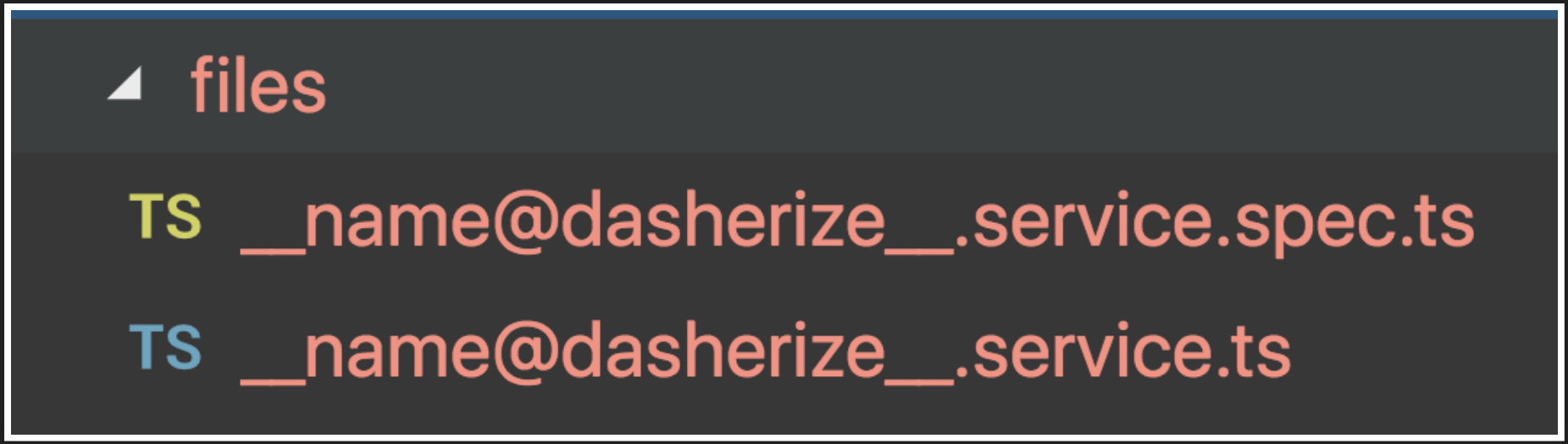
- define scaffold files
- define templates
- replace file name with command line args
- replace template content with commandl ine args

Scaffold a new blank project

```
schematics blank --name=my-template
```

define template file
you can place them where you want but your tsconfig.json is set to ignore /files

let's create some template files



```
export class <%= name %> {
  getData() {
    // fetch data here
  }
}
```

we are looking to replace `name`

```
<%= name %>
```

let's write some code

```
export function myTemplate(_options: any): Rule {
  _options.path = _options.path ? normalize(_options.path) : _
  const templateSource = apply(url("./files"), [
    filterTemplates(_options),
    template({
      ...stringUtils,
      ..._options
    }),
    move("/src/app/my-template")
    // what to do with the files in here
]);
  return chain([branchAndMerge(chain([mergeWith(templateSource
}
```

let's break it down

- **filterTemplates**, function that ensures we get the template files we want
- **template**, function that takes a template and content and merge them
- **move**, where we place the end result

if service is NOT specified as command line arg, then we don't include it else just filter out .bak files

```
function filterTemplates(options: any): Rule {
  if (!options.service) {
    return filter(
      path =>
      !path.match(/\.service\.ts$/) &&
      !path.match(/-item\.ts/) &&
      !path.match(/\.bak$/)
    );
  }
  return filter(path => !path.match(/\.bak$/));
}
```

`stringUtils` obj containing helper functions for our template, e.g `classify` `_options` obj containing data that we want to pass to our template, e.g `name`

```
import { classify, dasherize } from "@angular-devkit/core/src/

const stringUtils = { classify, dasherize };

template({
  ...stringUtils,
  ..._options
})
```

let's update our template

`classify` comes from `stringUtils`, ensures we get capital letter name `dasherize`, comes from `stringUtils`, ensures `ProductDetail` becomes `product-detail` `name` from _options

```
export class <%= classify(name) %> {
  getData() {
  // fetch data here
  }
}
```

looking at end result

Execute our schematic

```
npm run build && schematics .:my-template --name=ProductDetail --service=true --dry-run=false"
```

my-template

TS product-detail.service.spec.ts

TS product-detail.service.ts

```
export class ProductDetail {
  getData() {
    // fetch data here
  }
}
```

THE ANATOMY OF A TREE

```
tree.actions // Action[]
```

- is a list of type `Action`
- let's look at what an `Action` is

```
export declare type Action =
CreateFileAction |
OverwriteFileAction |
RenameFileAction |
DeleteFileAction;

export interface ActionBase {
    readonly id: number;
    readonly parent: number;
    readonly path: Path;
}
```

- different type of actions,
- `path` looks interesting
- `kind` is a property that says what type of CRUD

ok, an `Action` seems to represent our intent, how do we read from it?

```
tree.actions.forEach(action => {
  console.log("action kind", action.kind);
  console.log("action path", action.path);
  let fe = tree.get(action.path);
  if (fe) {
    console.log("action content", fe.content.toString("utf8"))
  }
});
```

ok so we can list all the things the tree is about to do, with its content

This is internals, it might change in the near future, still interesting though

UNDERSTANDING YOUR CODE

USING THE TYPESCRIPT COMPILER

We can send in our code into the Typescript compiler and it will give us a tree representation back

- read file
- convert file content into a `SourceFile`
- investigate your tree

very simple file

```
class DemoClass {
  constructor(otherDemo: DemoClass, param: string) {}
}
```

read, create source file

```
let node = ts.createSourceFile(
  'demo.ts',
  content,
  ts.ScriptTarget.Latest,
  true
);
```

let's try to print our nodes in a nice way

thank you Manfred Steyer for this function

```
function showTree(node: ts.Node, indent: string = '    '): voi
  console.log(indent + ts.SyntaxKind[node.kind]);

  if (node.getChildCount() === 0) {
      console.log(indent + '    Text: ' + node.getText());
  }

  for(let child of node.getChildren()) {
      showTree(child, indent + '    ');
  }
}

showTree(node);
```

```
SourceFile
    SyntaxList
        ClassDeclaration
            ClassKeyword
                Text: class
            Identifier
                Text: DemoClass
            FirstPunctuation
                Text: {
            SyntaxList
                Constructor
                    ConstructorKeyword
                        Text: constructor
                    OpenParenToken
                        Text: (
                    SyntaxList
                        Parameter
                            Identifier
                                Text: otherDemo
                            ColonToken
                                Text: :
                            TypeReference
                                Identifier
                                    Text: DemoClass
                        CommaToken
                            Text: ,
                        Parameter
                            Identifier
                                Text: param
                            ColonToken
                                Text: :
                            StringKeyword
                                Text: string
                    CloseParenToken
                        Text: )
                    Block
                        FirstPunctuation
                            Text: {
                        SyntaxList
                            Text:
                        CloseBraceToken
                            Text: }
```

TODO show tree content

What about changing the tree?

- we need to flatten the tree into a list of nodes, for convenience
- we need to find the correct position to do our change
- we need to create a `Change` object that says `path`, `position of change` and `with what content`
- lastly we need to call the following on the tree `beginUpdate`, to get a recorder `commitUpdate` to make the changes stick
- admire our work

gives us a flattened list of Nodes

```
const nodes = getSourceNodes(componentSource);
```

you can query nodes by type, find the type you need according to our AST tree

```
function getNodeByType(nodes: ts.Node[], type: ts.SyntaxKind)
  return nodes.find(n => n.kind === type)[0];
}
// e.g
const node = getNodeByType(
  nodes,
  ts.SyntaxKind.ConstructorKeyword
)
```

once you have the correct node, simply insert before/after your node

```
function createChange(path: any, what: string, pos: number) {
  return new InsertChange(path, pos, what);
}

createChange('path-to-file', 'insert this', node.pos+1)
```

interate our list of changes and `commit`/ `save`

```
function applyChanges(path: string, content: string, changes:
  tree.create(path, content);
  const exportRecorder = tree.beginUpdate(path);
  for (const change of changes) {
    if (change instanceof InsertChange) {
      exportRecorder.insertLeft(change.pos, change.toAdd);
    }
  }
  tree.commitUpdate(exportRecorder);
}
```

all together now :)

```
function myRule(options): Rule {
  return (tree: Tree, _context: SchematicContext) => {
    const componentSource = getTsSource( componentPath,
      componentContent
    );
    const nodes = getSourceNodes(componentSource);
    const node = getNodeByType( nodes,
      ts.SyntaxKind.ConstructorKeyword
    );
    const change = createChange( componentPath,
      'insert this',
      node.pos+1
    );
    applyChanges( componentPath,
      componentContent, [ change ], tree);
```

FURTHER READING

- Intro article to Schematics article by Hans from Angula
https://blog.angular.io/schematics-an-introduction-
dc1dfbc2a2b2
- Manfred Steyer, 5 blog post series,
https://www.softwarearchitekt.at/post/2017/10/29/ge
custom-code-with-the-angular-cli-and-schematics.as
- Manfred Steyer, free book on Schematics,
https://leanpub.com/angular-schematics
- Jorge Cano, article, https://medium.com/@jorgeucan
fist-angular-schematics-f711d70cb37c

SUMMARY

- Workflow tool
- Performs changes on a Tree
- Consist of Rules you apply on a Tree
- Can be used for scaffolding but also update code
- is the secret sauce behind Angular-CLI

THANK YOU