**📖 MANUAL PARA TORPES - Script Python Generador**

**🎯 ¿QUÉ HACE EL SCRIPT?**

**En palabras simples:** Toma datos de archivos YAML + plantillas Jinja2 → Genera configuraciones de equipos de red

**Es como una fábrica:**

[YAML: datos] + [Plantilla: molde] → [Configuración: producto final]

**🗂️ ESTRUCTURA DE ARCHIVOS**

TFG-Metodologia-PYMES/

├── scripts/

│ └── generar\_config.py ← EL SCRIPT PRINCIPAL (la fábrica)

├── global\_config.yaml ← Datos comunes a todo

├── sedes/

│ └── sede\_simple.yaml ← Datos específicos de sede

├── dispositivos/

│ └── router\_simple.yaml ← Datos específicos de dispositivo

├── plantillas/

│ ├── router\_simple.j2 ← Moldes para generar configuraciones

│ ├── cifrado\_modulo.j2

│ └── gestion\_router\_modulo.j2

└── configuraciones/

└── alcorcon/ ← Configuraciones generadas (salida)

**🔄 FLUJO PASO A PASO**

**1. USUARIO EJECUTA COMANDO**

python generar\_config.py --sede "alcorcon" --tipo-sede sede\_simple --dispositivo router\_simple

**2. SCRIPT TRADUCE PARÁMETROS**

# Lo que recibe el script:

sede\_nombre = "alcorcon" # ← Desde --sede

tipo\_sede = "sede\_simple" # ← Desde --tipo-sede

tipo\_dispositivo = "router\_simple" # ← Desde --dispositivo

**3. CARGAR ARCHIVOS YAML (función cargar\_configuraciones)**

# Carga 3 archivos:

global\_config.yaml # ← Datos comunes (VRFs, BGP, etc.)

sedes/sede\_simple.yaml # ← Datos de la sede (IPs, VLANs, etc.)

dispositivos/router\_simple.yaml # ← Datos del dispositivo (interfaces, etc.)

**4. PREPARAR JINJA2 (función configurar\_jinja)**

# Prepara el "motor" que convierte plantillas:

FileSystemLoader('plantillas') # ← Busca plantillas en carpeta plantillas/

# Registra funciones especiales para IPs

**5. VALIDAR DATOS (función validar\_configuracion)**

# Comprueba que todo esté bien:

- ¿Existen las redes IP definidas?

- ¿Si cifrado=true, hay configuración de cifrado?

- ¿Los datos tienen sentido?

**6. ELEGIR PLANTILLA (función determinar\_plantilla)**

# Decide qué plantilla usar:

if --modulo-cifrado:

plantilla = "cifrado\_modulo.j2"

elif --modulo-gestion:

if router: plantilla = "gestion\_router\_modulo.j2"

if switch: plantilla = "gestion\_switch\_modulo.j2"

else:

# Configuración completa

plantilla = "router\_simple.j2" # ← Usa el nombre del --dispositivo

**7. GENERAR CONFIGURACIÓN (función generar\_configuracion)**

# Mezcla datos YAML + plantilla Jinja2:

plantilla.render(

global\_config = datos\_global, # ← Del global\_config.yaml

sede = datos\_sede, # ← Del sede\_simple.yaml

direccionamiento = IPs\_de\_sede, # ← Del sede\_simple.yaml

interfaces = interfaces\_router # ← Del router\_simple.yaml

)

# RESULTADO: Configuración lista para copiar al router

**8. GUARDAR ARCHIVO (función guardar\_configuracion)**

# Guarda en:

configuraciones/alcorcon/router\_simple\_20250602\_1355.txt

**🧠 FUNCIONES PRINCIPALES EXPLICADAS**

**🏭 Clase GeneradorConfiguraciones**

class GeneradorConfiguraciones:

def \_\_init\_\_(self):

self.global\_config = None # ← Aquí guarda datos del global\_config.yaml

self.sede\_config = None # ← Aquí guarda datos del sede\_simple.yaml

self.dispositivo\_config = None # ← Aquí guarda datos del router\_simple.yaml

self.jinja\_env = None # ← Aquí guarda el "motor" Jinja2

**Es como una caja que va guardando todo lo que necesita para trabajar.**

**📂 cargar\_configuraciones()**

def cargar\_configuraciones(self, sede\_nombre, tipo\_sede, tipo\_dispositivo):

# 1. Lee global\_config.yaml → self.global\_config

# 2. Lee sedes/sede\_simple.yaml → self.sede\_config

# 3. Cambia nombre genérico por nombre real: "alcorcon"

# 4. Lee dispositivos/router\_simple.yaml → self.dispositivo\_config

**Es como abrir 3 cajones y sacar los papeles con datos.**

**⚙️ configurar\_jinja()**

def configurar\_jinja(self):

# 1. Crea el "motor" que lee plantillas .j2

# 2. Le dice: "busca plantillas en carpeta plantillas/"

# 3. Le enseña funciones especiales para IPs:

# - cidr\_to\_netmask: /24 → 255.255.255.0

# - next\_ip: 192.168.1.2 → 192.168.1.3

**Es como preparar una máquina que sabe mezclar recetas.**

**✅ validar\_configuracion()**

def validar\_configuracion(self):

# Revisa si todo tiene sentido:

if cifrado activo BUT sin datos de cifrado:

ERROR ❌

if IP mal formada:

ERROR ❌

# Si todo bien: OK ✅

**Es como un inspector de calidad.**

**🎯 determinar\_plantilla()**

def determinar\_plantilla(self, tipo\_dispositivo, modulos...):

if modulo\_cifrado:

return "cifrado\_modulo.j2" # ← Plantilla de cifrado

elif modulo\_gestion and router:

return "gestion\_router\_modulo.j2" # ← Plantilla gestión router

else:

return f"{tipo\_dispositivo}.j2" # ← router\_simple.j2

**Es como un cajero que decide qué molde usar según lo que pidas.**

**🏗️ generar\_configuracion()**

def generar\_configuracion(self):

# 1. Toma la plantilla elegida: router\_simple.j2

# 2. Prepara todos los datos en un "diccionario":

contexto = {

'global\_config': self.global\_config, # ← Datos globales

'sede': self.sede\_config['sede'], # ← Datos de sede

'direccionamiento': self.sede\_config['direccionamiento'], # ← IPs

'interfaces': self.dispositivo\_config['interfaces'] # ← Interfaces

}

# 3. MAGIA: plantilla + contexto = configuración final

configuracion = plantilla.render(\*\*contexto)

**Es la fábrica principal. Toma molde + ingredientes = producto final.**

**🔍 CÓMO FUNCIONA JINJA2**

**Plantilla router\_simple.j2:**

hostname {{ sede.nombre | replace(" ", "\_") }}\_ppal

!

{% for vrf\_name, vrf\_config in global\_config.segmentacion.vrfs.items() %}

ip vrf {{ vrf\_name }}

rd {{ vrf\_config.rd }}

{% endfor %}

**Datos YAML:**

sede:

nombre: "alcorcon"

global\_config:

segmentacion:

vrfs:

main:

rd: "65000:1"

gestion:

rd: "65000:99"

**Resultado final:**

hostname alcorcon\_ppal

!

ip vrf main

rd 65000:1

ip vrf gestion

rd 65000:99

**Jinja2 es como rellenar huecos en un formulario, pero automáticamente.**

**🎮 EJEMPLOS PRÁCTICOS**

**Ejemplo 1: Router Completo**

python generar\_config.py --sede "madrid" --tipo-sede sede\_simple --dispositivo router\_simple

**Lo que hace:**

1. Carga: global\_config.yaml + sede\_simple.yaml + router\_simple.yaml
2. Busca: plantillas/router\_simple.j2
3. Mezcla: datos + plantilla = configuración completa
4. Guarda: configuraciones/madrid/router\_simple\_FECHA.txt

**Ejemplo 2: Solo Módulo de Cifrado**

python generar\_config.py --sede "madrid" --tipo-sede sede\_simple --dispositivo router\_simple --modulo-cifrado

**Lo que hace:**

1. Carga: los mismos YAMLs
2. Busca: plantillas/cifrado\_modulo.j2 (¡diferente plantilla!)
3. Mezcla: datos + plantilla = solo comandos de cifrado
4. Guarda: configuraciones/madrid/router\_simple\_modulo\_cifrado\_FECHA.txt

**🚨 MANEJO DE ERRORES**

**Error: Archivo YAML no existe**

except FileNotFoundError as e:

print(f"[ERROR] Archivo no encontrado: {e}")

return False

**Error: YAML mal formado**

except yaml.YAMLError as e:

print(f"[ERROR] Error al procesar YAML: {e}")

return False

**Error: Plantilla no existe**

except Exception as e:

print(f"[ERROR] Error generando configuracion: {e}")

return None

**El script intenta ser amigable cuando algo sale mal.**

**🎛️ PARÁMETROS DEL SCRIPT**

parser.add\_argument('--sede', required=True) # ← Nombre de la sede

parser.add\_argument('--tipo-sede', required=True) # ← Tipo: sede\_simple

parser.add\_argument('--dispositivo', required=True) # ← Tipo: router\_simple

parser.add\_argument('--modulo-cifrado', action='store\_true') # ← Flag: true/false

**action='store\_true' significa:**

* Si usuario pone --modulo-cifrado → True
* Si NO lo pone → False

**💡 TRUCOS PARA ENTENDER EL CÓDIGO**

**1. self = "yo mismo"**

class GeneradorConfiguraciones:

def \_\_init\_\_(self):

self.global\_config = None # ← "Mi variable global\_config"

def cargar\_configuraciones(self):

self.global\_config = datos # ← "Guardo datos en MI variable"

**2. \*\*contexto = "expande diccionario"**

contexto = {'sede': 'madrid', 'vrf': 'main'}

plantilla.render(\*\*contexto)

# Es lo mismo que:

plantilla.render(sede='madrid', vrf='main')

**3. try/except = "intentar, si falla..."**

try:

archivo = open('datos.yaml') # ← Intentar

except FileNotFoundError:

print("Archivo no existe") # ← Si falla, hacer esto

**🎯 RESUMEN FINAL**

**El script es como una fábrica automática:**

1. **ENTRADA:** Parámetros del usuario (--sede, --dispositivo, etc.)
2. **MATERIAS PRIMAS:** Archivos YAML con datos
3. **MOLDES:** Plantillas Jinja2
4. **PROCESO:** Mezcla datos + plantilla
5. **SALIDA:** Configuración lista para el equipo

**Y todo está organizado en funciones que hacen una cosa específica:**

* Una función carga datos
* Otra valida
* Otra genera
* Otra guarda

**¡Como una cadena de montaje! 🏭**