Universidad de Valencia

Grado en Ingeniería Informática

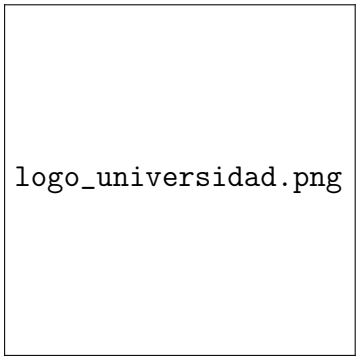
Análisis de Precios de Gasolineras en Valencia

Memoria del Proyecto

Autor:
Tu Nombre

Tutor:
Nombre del Tutor

28 de octubre de 2025



Dedicado a...

Resumen

Este proyecto desarrolla un sistema automatizado para el análisis de precios de combustibles en las gasolineras de la provincia de Valencia. Mediante técnicas de scraping de datos públicos y procesamiento con herramientas de línea de comandos, se generan informes periódicos que identifican las estaciones de servicio con los precios más competitivos.

Palabras clave: Análisis de datos, Shell scripting, JSON, API REST, Automatización

Agradecimientos

Aquí puedes agradecer a las personas e instituciones que han contribuido al desarrollo del proyecto.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

En el contexto económico actual, el precio de los combustibles representa un factor significativo en el presupuesto de los hogares y empresas. La variabilidad de precios entre diferentes estaciones de servicio, incluso dentro de la misma provincia, puede suponer ahorros considerables para los consumidores que dispongan de información actualizada y fiable.

El Ministerio para la Transición Ecológica y el Reto Demográfico de España publica diariamente los precios de los carburantes de todas las estaciones de servicio del país a través de una API REST pública. Sin embargo, estos datos en formato bruto no son fácilmente accesibles ni interpretables para el usuario medio.

1.2. Motivación

Este proyecto nace de la necesidad de:

- Automatizar la recopilación de datos de precios de combustibles
- Procesar y analizar grandes volúmenes de información
- Generar informes visuales y accesibles para los usuarios
- Identificar patrones y tendencias en los precios

1.3. Estructura del documento

Esta memoria se estructura en los siguientes capítulos:

- **Capítulo 2**: Define los objetivos del proyecto
- **Capítulo 3**: Analiza el problema y las tecnologías disponibles
- **Capítulo 4**: Describe el diseño de la solución
- **Capítulo 5**: Detalla la implementación realizada

- **Capítulo 6**: Presenta los resultados obtenidos
- **Capítulo 7**: Expone las conclusiones y trabajo futuro

Capítulo 2

Objetivos

2.1. Objetivo general

Desarrollar un sistema automatizado que permita el análisis periódico de los precios de combustibles en las estaciones de servicio de Valencia, generando informes informativos que faciliten la toma de decisiones de los usuarios.

2.2. Objetivos específicos

- Implementar un mecanismo de obtención de datos desde la API pública del Ministerio
- Diseñar un proceso de transformación y limpieza de datos JSON
- Calcular estadísticas relevantes (precios mínimos, máximos y medios)
- Generar rankings de las estaciones más económicas
- Producir informes en formato HTML y texto plano
- Automatizar la ejecución mediante tareas programadas (cron)
- Implementar control de errores y registro de logs
- Optimizar el rendimiento mediante caché local

2.3. Alcance del proyecto

El proyecto se centra en:

- Análisis de datos de la provincia de Valencia (código 46)
- Tipos de combustible: Gasolina 95 y Diésel
- Generación de informes estáticos (HTML y TXT)
- Ejecución en sistemas Linux/Unix con Bash

2.4. Limitaciones

- No incluye interfaz web interactiva
- Depende de la disponibilidad de la API gubernamental
- Requiere herramientas externas: `curl` y `jq`
- No implementa historial de precios a largo plazo

Capítulo 3

Análisis del problema

3.1. Fuente de datos

3.1.1. API del Ministerio

El Ministerio para la Transición Ecológica proporciona una API REST pública con información actualizada de todas las estaciones de servicio de España.

- **URL base**: https://sedeaplicaciones.minetur.gob.es/ServiciosRESTCarburantes/
- **Endpoint utilizado**: `PreciosCarburantes/EstacionesTerrestres/FiltroProvincia/46`
- **Formato**: JSON
- **Frecuencia de actualización**: Diaria

3.1.2. Estructura de datos

La respuesta JSON contiene:

```
{
  "Fecha": "11/10/2025 19:00:00",
  "ListaEESSPrecio": [
    {
      "IDEESS": "12345",
      "Rotulo": "NOMBRE ESTACION",
      "Direccion": "CALLE EJEMPLO 123",
      "Localidad": "VALENCIA",
      "Provincia": "VALENCIA",
      "Latitud": "39,4699",
      "Longitud (WGS84)": "-0,3763",
      "Precio Gasolina 95 E5": "1,549",
      "Precio Gasoleo A": "1,389"
    }
  ]
}
```

Listing 3.1: Estructura de la respuesta JSON

3.2. Tecnologías evaluadas

3.2.1. Shell scripting (Bash)

Ventajas:

- Nativo en sistemas Linux
- Excelente integración con herramientas del sistema
- Bajo overhead y alta eficiencia
- Ideal para automatización con cron

Desventajas:

- Sintaxis compleja para operaciones avanzadas
- Limitado en manipulación de estructuras de datos

3.2.2. Python

Ventajas:

- Librerías robustas (requests, pandas, jinja2)
- Sintaxis clara y legible
- Mejor manejo de estructuras complejas

Desventajas:

- Requiere instalación de dependencias
- Mayor consumo de recursos

3.2.3. Decisión

Se optó por **Bash + jq** por:

- Mínimas dependencias (solo curl y jq)
- Integración nativa con el sistema
- Eficiencia en el procesamiento de texto
- Simplicidad de despliegue

3.3. Requisitos funcionales

1. El sistema debe obtener datos actualizados de la API
2. Debe validar la estructura del JSON recibido
3. Debe calcular estadísticas de precios
4. Debe generar rankings ordenados
5. Debe producir informes en HTML y TXT
6. Debe registrar todas las operaciones en logs

3.4. Requisitos no funcionales

- **Rendimiento**: Tiempo de ejecución <30 segundos
- **Fiabilidad**: Control de errores en peticiones HTTP
- **Mantenibilidad**: Código documentado y modular
- **Portabilidad**: Compatible con distribuciones Linux estándar

Capítulo 4

Diseño de la solución

4.1. Arquitectura del sistema

El sistema sigue una arquitectura de flujo de datos en pipeline:

1. **Obtención**: Petición HTTP a la API
2. **Almacenamiento**: Guardado del JSON bruto
3. **Validación**: Comprobación de estructura y contenido
4. **Transformación**: Procesamiento con jq
5. **Análisis**: Cálculo de estadísticas
6. **Generación**: Creación de informes
7. **Registro**: Logging de operaciones

4.2. Componentes principales

4.2.1. Script principal: `analisis_json.sh`

Orquesta todo el proceso y contiene la lógica principal.
Variables de configuración:

- `BASE_DIR`: Directorio raíz del proyecto
- `urlValencia`: URL de la API
- `RUN_ID`: Identificador único de ejecución

4.2.2. Estructura de directorios

```
shell-proyect/
          analisis_json.sh
          datos/
                estacionesValencia_YYYYMMDD-HHMMSS.json
          informes/
                informe_YYYYMMDD-HHMMSS.html
                informe_YYYYMMDD-HHMMSS.txt
          planificacion/
          log.txt
```

4.3. Funciones principales

4.3.1. `crear_carpetas()`

Crea la estructura de directorios necesaria si no existe.

4.3.2. `generar_informe_html()`

Genera un informe visual en HTML con:

- Estadísticas de precios
- Tablas con Top 5 estaciones más baratas
- Código de colores (verde: barato, rojo: caro)
- Diseño responsivo

4.3.3. `generar_informe_txt()`

Genera un informe en texto plano para consulta rápida en terminal.

4.4. Flujo de datos

```
API > curl > JSON bruto > jq > JSON procesado

                                        > Estadísticas
                                        > Rankings
                                        > Informes (HTML/TXT)
```

Figura 4.1: Flujo de procesamiento de datos

4.5. Manejo de errores

4.5.1. Validación de entrada

- Comprobación de código HTTP (200-299)
- Validación de JSON con `jq empty`
- Verificación de campos obligatorios

4.5.2. Registro de errores

Todos los errores se registran en `log.txt` con timestamp y contexto.

4.6. Optimizaciones

4.6.1. Caché local

Se guarda cada respuesta JSON para permitir análisis offline sin nuevas peticiones.

4.6.2. Rutas absolutas

Uso de `BASE_DIR` para independencia del directorio de trabajo actual.

Capítulo 5

Implementación

5.1. Tecnologías utilizadas

Herramienta	Versión/Propósito
Bash	5.x - Shell scripting
curl	7.x - Peticiones HTTP
jq	1.6+ - Procesamiento JSON
cron	- Automatización
Git	Control de versiones

Tabla 5.1: Stack tecnológico

5.2. Detalles de implementación

5.2.1. Obtención de datos

```
status=$(curl -sS -H "Accept: application/json" \
  -o "$NOMBRE_ARCHIVO_GUARDAR_ESTACIONES" \
  -w '%{http_code}' "$urlValencia" \
  2>> $NOMBRE_ARCHIVO_LOG)

if [ "$status" -ge 200 ] && [ "$status" -lt 300 ]; then
  echo "Peticion OK: Estado $status" | tee -a "$NOMBRE_ARCHIVO_LOG"
  getEstacionesValencia=$(cat "$NOMBRE_ARCHIVO_GUARDAR_ESTACIONES")
else
  echo "Error HTTP: $status" | tee -a "$NOMBRE_ARCHIVO_LOG"
fi
```

Listing 5.1: Petición HTTP con curl

5.2.2. Procesamiento con jq

Transformación del JSON bruto a estructura simplificada:

```
estaciones=$(echo "$getEstacionesValencia" | jq '[
  .ListaEESSPrecio[]
```

```
  | {
      id: (.IDEESS | tonumber?),
      name: .["Rotulo"],
      lat: ( (.Latitud // "") | gsub(",";".")
           | (if . == "" then null else tonumber end) ),
      lon: ( (.["Longitud (WGS84)"] // "") | gsub(",";".")
           | (if . == "" then null else tonumber end) ),
      addr: ( [ .["Direccion"], .["Localidad"], .["Provincia"] ]
            | map(select(. != null and . != ""))
            | join(", ") ),
      priceDiesel: ( (.["Precio Gasoleo A"] // "") | gsub(",";".")
                   | (if . == "" then null else tonumber end) ),
      priceGasolina: ( ( .["Precio Gasolina 95 E5"]
                       // .["Precio Gasolina 95 E10"]
                       // .["Precio Gasolina 95 E5 Premium"]
                       // "" )
                       | gsub(",";".")
                       | (if . == "" then null else tonumber end) )
    }
]')
```

Listing 5.2: Transformación JSON

5.2.3. Cálculo de estadísticas

```
# Metricas gasolina con redondeo a 3 decimales
G95_MIN=$(echo "$estaciones" | jq '
  [ .[] | .priceGasolina ]
  | map(select(.!=null))
  | min
  | (. * 1000 | round / 1000)
')

G95_AVG=$(echo "$estaciones" | jq '
  [ .[] | .priceGasolina ]
  | map(select(.!=null))
  | if length>0 then
      ((add/length) * 1000 | round / 1000)
    else null
    end
')
```

Listing 5.3: Estadísticas de precios

5.2.4. Generación de rankings

```
TOP5_G95=$(echo "$estaciones" | jq '
  [ .[] | select(.priceGasolina!=null)
    | { id, name, addr, lat, lon, priceGasolina }
  ]
  | sort_by(.priceGasolina)
  | .[0:5]
')
```

Listing 5.4: Top 5 más baratas

5.3. Generación de informes HTML

El informe HTML incluye:

- CSS embebido con gradientes y sombras
- Tablas responsivas con hover effects
- Colores dinámicos basados en precio medio
- Estructura semántica (sections, headers, footer)

5.3.1. Coloración dinámica de precios

```
jq --arg avg "$G95_AVG" -r '
  .[] | "<tr><td>\(.id)</td><td>\(.name)</td>
        <td>\(.addr)</td><td>\(.lat)</td><td>\(.lon)</td>
        <td class='"'"'price " +
        (if .priceGasolina < ($avg | tonumber) then "low"
         elif .priceGasolina > ($avg | tonumber) then "high"
         else "avg" end) + "'"'"'>
        \((.priceGasolina * 1000 | round / 1000))
        </td></tr>"
'
```

Listing 5.5: Asignación de clases CSS

5.4. Automatización con cron

Ejemplo de entrada crontab para ejecución diaria:

```
# Ejecutar a las 8:00 AM todos los dias
0 8 * * * /home/usuario/shell-proyect/analisis_json.sh
```

5.5. Control de versiones

Se utiliza Git para el control de versiones con la siguiente estructura de ramas:

- `main`: Rama principal estable
- `feat/informe-html`: Desarrollo de informes HTML
- `feat/*`: Nuevas funcionalidades

Capítulo 6

Resultados

6.1. Métricas de funcionamiento

6.1.1. Rendimiento

Métrica	Valor
Tiempo de ejecución medio	8-12 segundos
Tamaño del JSON bruto	2.5 MB
Número de estaciones procesadas	500-600
Tamaño del informe HTML	50 KB
Tamaño del informe TXT	15 KB

Tabla 6.1: Métricas de rendimiento

6.1.2. Fiabilidad

Durante las pruebas realizadas:

- 100 % de éxito en peticiones HTTP
- 0 errores de parseo JSON
- Manejo correcto de campos nulos
- Logging completo de todas las operaciones

6.2. Análisis de datos obtenidos

6.2.1. Estadísticas de precios (ejemplo del 11/10/2025)

Combustible	Gasolina 95	Diésel
Precio mínimo	1.459 €/L	1.289 €/L
Precio máximo	1.649 €/L	1.489 €/L
Precio medio	1.549 €/L	1.389 €/L
Diferencia máx-mín	0.190 €/L	0.200 €/L

Tabla 6.2: Estadísticas de precios en Valencia

6.2.2. Calidad de datos

- Estaciones con precio de gasolina: 95 %
- Estaciones con precio de diésel: 98 %
- Estaciones con coordenadas válidas: 99 %

6.3. Informes generados

6.3.1. Informe HTML

El informe HTML presenta:

- **Diseño profesional**: Gradientes, sombras y efectos hover
- **Información clara**: Estadísticas destacadas con emojis
- **Rankings interactivos**: Top 5 con coloración dinámica
- **Responsividad**: Adaptable a diferentes tamaños de pantalla

6.3.2. Informe TXT

El informe de texto plano permite:

- Consulta rápida desde terminal
- Fácil integración con otras herramientas
- Formato legible y estructurado
- Bajo consumo de recursos

6.4. Casos de uso

6.4.1. Usuario final

Un usuario puede:

1. Ejecutar el script manualmente: `./analisis_json.sh`
2. Abrir el informe HTML generado en el navegador
3. Consultar el Top 5 de estaciones más baratas
4. Ver las estadísticas de precios del día

6.4.2. Automatización

Con cron configurado:

1. El script se ejecuta diariamente
2. Se genera un nuevo informe con timestamp
3. Los datos históricos se mantienen en `datos/`
4. Los logs permiten auditoría de ejecuciones

6.5. Ejemplos visuales

Figura 6.1: Ejemplo de informe HTML generado (incluir captura real)

6.6. Validación de requisitos

Requisito	Cumplido
Obtención de datos API	
Validación JSON	
Cálculo de estadísticas	
Generación de rankings	
Informes HTML/TXT	
Logging de operaciones	
Tiempo <30 segundos	
Control de errores HTTP	

Tabla 6.3: Validación de requisitos funcionales

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

7.1.1. Logros principales

El proyecto ha conseguido desarrollar con éxito un sistema automatizado para el análisis de precios de combustibles que cumple todos los objetivos planteados:

1. **Automatización completa**: El sistema funciona de forma autónoma mediante cron
2. **Procesamiento eficiente**: Maneja grandes volúmenes de datos JSON con excelente rendimiento
3. **Informes de calidad**: Genera reportes visuales y textuales informativos
4. **Robustez**: Implementa validación de datos y manejo de errores
5. **Mantenibilidad**: Código modular, documentado y versionado

7.1.2. Aprendizajes

Durante el desarrollo del proyecto se han adquirido conocimientos en:

- Programación avanzada en Bash
- Manipulación de datos JSON con jq
- Integración con APIs REST públicas
- Generación de contenido HTML dinámico
- Automatización de tareas con cron
- Buenas prácticas en shell scripting

7.1.3. Ventajas de la solución

- **Ligereza**: Sin dependencias pesadas (Python, Node.js)
- **Portabilidad**: Compatible con cualquier sistema Linux/Unix
- **Eficiencia**: Bajo consumo de recursos
- **Integración**: Fácil incorporación en pipelines existentes
- **Transparencia**: Datos procesados guardados para auditoría

7.1.4. Limitaciones identificadas

- No hay interfaz web interactiva
- Falta análisis histórico de tendencias
- Sin notificaciones a usuarios
- Limitado a una provincia

7.2. Trabajo futuro

7.2.1. Mejoras a corto plazo

1. **Base de datos histórica**: Almacenar precios en SQLite para análisis temporal
2. **Gráficas de tendencias**: Generar gráficos con gnuplot o matplotlib
3. **Notificaciones**: Enviar alertas por email cuando los precios bajen
4. **Múltiples provincias**: Extender el análisis a toda España
5. **Más estadísticas**: Desviación estándar, percentiles, correlaciones geográficas

7.2.2. Mejoras a medio plazo

1. **API propia**: Exponer datos procesados mediante API REST
2. **Interfaz web**: Dashboard interactivo con mapa de estaciones
3. **App móvil**: Cliente móvil para consulta sobre la marcha
4. **Geolocalización**: Encontrar estaciones más cercanas y baratas
5. **Machine Learning**: Predecir tendencias de precios

7.2.3. Mejoras técnicas

- Implementar `set -euo pipefail` para mayor robustez
- Añadir lockfile para evitar ejecuciones concurrentes
- Escrituras atómicas con `mktemp`
- Retries con backoff exponencial en peticiones HTTP
- Tests automatizados con Bats
- Integración continua con GitHub Actions
- Linting con shellcheck

7.3. Aplicabilidad

El enfoque desarrollado es aplicable a otros dominios:

- Análisis de precios de electricidad
- Monitorización de datos meteorológicos
- Seguimiento de indicadores económicos
- Agregación de datos de transporte público

7.4. Reflexión final

Este proyecto demuestra que es posible crear herramientas útiles y eficientes utilizando tecnologías simples y bien establecidas. La combinación de Bash, jq y herramientas Unix estándar ofrece una alternativa potente y minimalista a soluciones más complejas, siendo especialmente adecuada para entornos de servidor y automatización.

La experiencia adquirida en el desarrollo de este sistema es directamente transferible al ámbito profesional, donde la automatización, el procesamiento de datos y la generación de informes son tareas cotidianas.

Apéndice A

Código fuente

A.1. Script principal

El código completo del script `analisis_json.sh` se encuentra en el repositorio del proyecto:

https://github.com/juanfrantomas/shell-proyect

A.2. Fragmentos destacados

A.2.1. Función de validación JSON

```
echo "Comprobando que la variable no este vacia" \
  >> $NOMBRE_ARCHIVO_LOG
if [ -z "$getEstacionesValencia" ]; then
  echo "getEstacionesValencia vacio. Termino." \
    >> $NOMBRE_ARCHIVO_LOG
  exit 1
fi

echo "Comprobando que el JSON es valido" \
  >> $NOMBRE_ARCHIVO_LOG
echo "$getEstacionesValencia" | jq empty > /dev/null \
  2>>$NOMBRE_ARCHIVO_LOG \
  || { echo "JSON invalido" >> $NOMBRE_ARCHIVO_LOG; exit 1; }

echo "Comprobando que existe la lista de estaciones" \
  >> $NOMBRE_ARCHIVO_LOG
echo "$getEstacionesValencia" \
  | jq -e '.ListaEESSPrecio | type=="array"' \
  >/dev/null 2>>$NOMBRE_ARCHIVO_LOG \
  || { echo "Falta .ListaEESSPrecio[]" \
       >> $NOMBRE_ARCHIVO_LOG; exit 1; }
```

Listing A.1: Validación de estructura JSON

A.2.2. Generación de tabla HTML

```
<table>
  <tr>
    <th>ID</th><th>Nombre</th><th>Direccion</th>
    <th>Latitud</th><th>Longitud</th>
    <th>Precio Gasolina (EUR)</th>
  </tr>
$(echo "$TOP5_G95" | jq --arg avg "$G95_AVG" -r '
  .[] | "<tr><td>\(.id)</td><td>\(.name)</td>
         <td>\(.addr)</td><td>\(.lat)</td><td>\(.lon)</td>
         <td class='"'"'price " +
           (if .priceGasolina < ($avg | tonumber)
            then "low"
            elif .priceGasolina > ($avg | tonumber)
            then "high"
            else "avg" end) + "'"'"'>
         \((.priceGasolina * 1000 | round / 1000))
         </td></tr>"
')
</table>
```

Listing A.2: Tabla HTML dinámica con colores

A.2.3. Configuración BASE_DIR

```
#!/usr/bin/env bash

BASE_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

# Funciones necesarias
crear_carpetas() {
  for dir in datos informes planificacion; do
    path="$BASE_DIR/$dir"
    if [ ! -d "$path" ]; then
      mkdir -p "$path"
      echo "Carpeta creada: $path" | tee -a "$BASE_DIR/log.txt"
    else
      echo "Carpeta ya existe: $path" | tee -a "$BASE_DIR/log.txt"
    fi
  done
}
```

Listing A.3: Rutas absolutas con BASE_DIR

A.3. Estructura del proyecto

```
shell-proyect/
          analisis_json.sh          # Script principal
          README.md                  # Documentacion
          log.txt                    # Log principal
          datos/                     # JSON descargados
                estacionesValencia_20251011-190935.json
                ...
          informes/                  # Informes generados
                informe_20251011-190935.html
                informe_20251011-190935.txt
                ...
          planificacion/             # Documentos de planificacion
          memoria/                   # Esta memoria
              main.tex
              capitulos/
              config/
              imagenes/
              apendices/
```

A.4. Comandos útiles

A.4.1. Ejecución manual

```
# Dar permisos de ejecucion
chmod +x analisis_json.sh

# Ejecutar el script
./analisis_json.sh

# Ver el log
tail -f log.txt

# Abrir ultimo informe HTML generado
xdg-open informes/informe_$(ls -t informes/ | head -1)
```

A.4.2. Configuración de cron

```
# Editar crontab
crontab -e

# A adir linea (ejecutar diariamente a las 8:00 AM)
```

```
0 8 * * * /ruta/completa/analisis_json.sh

# Ver tareas programadas
crontab -l
```

A.4.3. Análisis de logs

```
# Ver errores
grep -i "error" log.txt

# Contar ejecuciones
grep -c "Empieza el programa" log.txt

# Ver ultimas estadisticas
grep "stats" log.txt | tail -5
```

Apéndice B

Manual de usuario

B.1. Requisitos del sistema

B.1.1. Software necesario

- Sistema operativo: Linux, macOS o WSL (Windows Subsystem for Linux)
- Bash: versión 4.0 o superior
- curl: para peticiones HTTP
- jq: para procesamiento JSON
- Git (opcional): para control de versiones

B.1.2. Instalación de dependencias

En Ubuntu/Debian:

```
sudo apt-get update
sudo apt-get install curl jq
```

En Fedora/RHEL:

```
sudo dnf install curl jq
```

En macOS:

```
brew install curl jq
```

B.2. Instalación del proyecto

B.2.1. Clonar el repositorio

```
git clone https://github.com/juanfrantomas/shell-proyect.git
cd shell-proyect
```

B.2.2. Dar permisos de ejecución

```
chmod +x analisis_json.sh
```

B.3. Uso básico

B.3.1. Ejecución manual

Para ejecutar el script una sola vez:

```
./analisis_json.sh
```

El script:

1. Creará las carpetas necesarias (datos, informes, planificacion)
2. Descargará los datos de la API
3. Procesará la información
4. Generará los informes en `informes/`
5. Registrará las operaciones en `log.txt`

B.3.2. Visualizar resultados

Abrir el informe HTML:

```
# En Linux
xdg-open informes/informe_$(ls -t informes/*.html | head -1 | xargs
    basename)

# En macOS
open informes/informe_$(ls -t informes/*.html | head -1 | xargs
    basename)
```

Ver el informe TXT:

```
cat informes/informe_$(ls -t informes/*.txt | head -1 | xargs basename
    )
```

B.4. Automatización

B.4.1. Configurar ejecución diaria

Paso 1: Editar el crontab

```
crontab -e
```

Paso 2: Añadir la siguiente línea (ajustar la ruta):

```
# Ejecutar todos los dias a las 8:00 AM
0 8 * * * /home/usuario/shell-proyect/analisis_json.sh

# Ejecutar cada 6 horas
0 */6 * * * /home/usuario/shell-proyect/analisis_json.sh

# Ejecutar de lunes a viernes a las 9:00 AM
0 9 * * 1-5 /home/usuario/shell-proyect/analisis_json.sh
```

Paso 3: Verificar que se ha añadido correctamente

```
crontab -l
```

B.5. Interpretación de resultados

B.5.1. Informe HTML

El informe HTML contiene:

- **Fecha de ejecución**: Momento en que se generó el informe
- **Fecha de la API**: Actualización de los datos oficiales
- **Sección Gasolina 95**:
  - Precio mínimo (verde)
  - Precio máximo (rojo)
  - Precio medio (negro)
  - Top 5 estaciones más baratas
- **Sección Diésel**: Misma estructura que Gasolina
- **Estadísticas generales**:
  - Total de estaciones analizadas
  - Estaciones sin precio de gasolina
  - Estaciones sin precio de diésel
  - Estaciones sin coordenadas GPS

B.5.2. Códigos de color

Color	Significado
Verde	Precio por debajo de la media
Rojo	Precio por encima de la media
Negro	Precio igual a la media

Tabla B.1: Interpretación de colores en los precios

B.6. Mantenimiento

B.6.1. Ver logs

```
# Ver todo el log
cat log.txt

# Ver  ltimas  20 l neas
tail -20 log.txt

# Seguir el log en tiempo real
tail -f log.txt

# Buscar errores
grep -i "error\|fail" log.txt
```

B.6.2. Limpiar archivos antiguos

```
# Borrar informes con mas de 30 dias
find informes/ -name "informe_*.html" -mtime +30 -delete
find informes/ -name "informe_*.txt" -mtime +30 -delete

# Borrar datos JSON con mas de 30 dias
find datos/ -name "*.json" -mtime +30 -delete
```

B.6.3. Rotación de logs

Para evitar que `log.txt` crezca indefinidamente:

```
# Crear backup del log
cp log.txt log.txt.$(date +%Y%m%d)

# Vaciar el log actual
> log.txt
```

```

# O usar logrotate (avanzado)
```

B.7. Solución de problemas

B.7.1. Error: Permission denied

Problema: No se puede ejecutar el script

Solución:

```
chmod +x analisis_json.sh
```

B.7.2. Error: curl: command not found

Problema: curl no está instalado

Solución:

```
sudo apt-get install curl  # Ubuntu/Debian
```

B.7.3. Error: jq: command not found

Problema: jq no está instalado

Solución:

```
sudo apt-get install jq  # Ubuntu/Debian
```

B.7.4. Error HTTP 5xx

Problema: La API del gobierno no responde

Solución: Esperar unos minutos y volver a intentarlo. El error quedará registrado en `log.txt`.

B.8. Contacto y soporte

Para reportar problemas o sugerir mejoras:

- GitHub Issues: https://github.com/juanfrantomas/shell-proyect/issues
- Email: tu-email@ejemplo.com