Alias Inteligentes (v3.1) para ecosistemas DevSecOps

Bitacora | Vol. 1

mercedev.es — 2026-05-04 | Fase 8.4 (Identidad y Autoridad)

Actualización v1.0: Conciencia de Contexto (Context-Aware)

El Desafío (Síntoma): Al ejecutar el orquestador local (`merci total`) desde la terminal en un nuevo repositorio derivado (Merci Boilerplate), el sistema auditaba por error el proyecto padre original (`mercedev.es`) en lugar del directorio activo. Esto ocurrió porque se habían configurado alias de terminal (`~/.zshrc`) con rutas absolutas. Al ejecutar el script del proyecto padre, Python utilizaba `Path(__file__).resolve().parents` para descubrir su ubicación física, auditando la carpeta donde residía el archivo y no el directorio actual de trabajo.

La Maniobra (Lógica): Se descartó la práctica de definir alias estáticos y se reemplazaron por una función Bash inteligente sensible al contexto.

```
 # Motor Merci - Ejecutor Inteligente inicial
merci() {
    if [ -f "scripts/merci/merci-$1.py" ]; then
        python3 "scripts/merci/merci-$1.py"
    else
        echo "🛡️ [Merci Error] No estás en la raíz de un proyecto Merci o el comando '$1' no existe."
    fi
}
```

El Aprendizaje (Fantasmas en RAM): Al posicionarse en la raíz de cualquier repositorio Merci, la función busca la ruta relativa e invoca al script correcto. Se descubrió que la sesión activa de la terminal mantiene los alias vivos en la memoria volátil; para purgar el estado tras un cambio en el perfil, es obligatorio ejecutar `unalias` o reiniciar la sesión.

Actualización v2.0: Parámetros Infinitos y Recarga en Caliente

El Desafío (Síntoma): La función original era rígida y no permitía pasar argumentos adicionales (flags como `--verbose` , `-v` o rutas de archivos) a los scripts subyacentes, limitando el uso de herramientas como el orquestador de backups o el publicador Headless (Sistema de Gestión de Contenidos desacoplado).

La Maniobra (Lógica): Se actualizó la función inyectando la variable de expansión `${@:2}` de Zsh/Bash, que captura todos los argumentos a partir del segundo.

```
merci() {
    if [ -f "scripts/merci/merci-$1.py" ]; then
        # Ejecuta el script pasándole todos los argumentos extra
        python3 "scripts/merci/merci-$1.py" "${@:2}"
    else
        echo "🛡️ [Merci Error] No estás en la raíz o el comando no
existe."
    fi
}
```

El Aprendizaje (Recarga de Terminal): Al modificar el archivo `~/.zshrc`, los cambios no se aplican mágicamente a las terminales abiertas. Es obligatorio ejecutar `source ~/.zshrc` para inyectar la configuración en caliente.

Actualización v3.0: Enrutamiento al Entorno Virtual (Runtime vs Buildtime)

El Desafío (Síntoma): La ejecución de herramientas locales fallaba si se olvidaba activar el entorno virtual de Python (`source .venv/bin/activate`), generando fricción operativa. Se debatió si eliminar dependencias de compilación para evadir el uso de entornos virtuales, lo que degradaría la arquitectura.

La Maniobra (Lógica): Se clarificó la regla arquitectónica: la política de "0 dependencias" aplica estrictamente al entorno de ejecución (Runtime en navegador), no a las herramientas de construcción (Buildtime). Se actualizó la función para apuntar explícitamente al binario aislado del entorno:

```
merci() {
    if [ -f "scripts/merci/merci-$1.py" ]; then
        # Actualización v3.0: Enrutamiento directo al binario del
entorno virtual (.venv)
        # Permite ejecutar herramientas sin necesidad de activar el
entorno manualmente.
        .venv/bin/python "scripts/merci/merci-$1.py" "${@:2}"
    else
```

```
        echo "🛡️ [Merci Error] No estás en la raíz o el comando no
existe."
    fi
}
```

El Aprendizaje (Aislamiento y DX): Al usar la ruta explícita del binario (`.venv/bin/python`), el sistema operativo resuelve automáticamente las librerías instaladas en ese entorno aislado sin necesidad de que la sesión de la terminal esté "activada", logrando fricción cero.

Actualización v3.1: Fallback Dinámico a Python Global (Fail Gracefully)

El Desafío (Síntoma): Al instanciar un nuevo clon del Boilerplate, la ejecución de `merci init` fallaba porque el entorno virtual aún no existía, rompiendo el flujo inicial de instalación.

La Maniobra (Lógica): Se inyectó un condicional dentro del enrutador para aplicar una degradación elegante (*Fail Gracefully*):

```
merci() {
    if [ -f "scripts/merci/merci-$1.py" ]; then
        if [ -f ".venv/bin/python" ]; then
            # Usa el entorno virtual si existe (fricción cero)
            .venv/bin/python "scripts/merci/merci-$1.py" "${@:2}"
        else
            # Fallback al Python global para clones limpios
            python3 "scripts/merci/merci-$1.py" "${@:2}"
        fi
    else
        echo "🛡️ [Merci Error] No estás en la raíz de un proyecto Merci
o el comando '$1' no existe."
    fi
}
```

El Aprendizaje (Adaptabilidad): Dotar al alias de la inteligencia para sobrevivir en entornos limpios y hacer un fallback dinámico al binario global es un paso

fundamental hacia el verdadero "Cero Mantenimiento" y una experiencia "Out-of-the-Box".

Fuentes / **Bibliografía:** * Asistencia mediante IA sobre el comportamiento de `resolve().parents` en Python. * Documentación POSIX sobre variables de expansión de shell (`$@`).