Informe de Diseño

Administrador ligero de respaldos y monitoreo (backup_admin.sh)

Autor: Willy Cuellar 23182

1. Resumen

Este documento describe la lógica y el diseño *top–down* del script backup_admin.sh, un administrador ligero de respaldos y monitoreo probado en Rocky Linux. El programa ofrece un menú interactivo para: (1) gestionar fuentes a respaldar, (2) seleccionar el destino, (3) ejecutar un respaldo incremental con preferencia por rsync (y *fallback* a cp -a), (4) consultar el estado del último respaldo, (5) activar un monitor de logs con umbrales/ventanas y alertas (stub o Telegram). Incluye además demos de I/O (§6) y diagnóstico con tee (§7).

2. Objetivos de diseño

- Robustez: validaciones de existencia/lectura de fuentes, espacio disponible, y permisos de escritura en destino.
- Trazabilidad: /backups/backup.log con *timestamp* y mensajes consistentes.
- Seguridad de prueba: modo -dry-run que simula todo sin efectuar cambios.
- Extensibilidad: funciones modulares, parámetros por CLI y configuración en ~/.backup_admin.conf.

3. Arquitectura *top–down*

El flujo parte en main(), que asegura estructura base, carga configuración, parsea flags y enruta a menú o a modo monitor:

```
main()
  ensure_base()      # crea ~/backups, log y lista de fuentes si no existen
  load_conf()        # lee DEST y credenciales de Telegram (~/.backup_admin.conf)
  parse_args(...)    # flags: --verbose, --dry-run, --monitor, etc.
  { MONITOR ? monitor_loop(...) : menu_principal() }
```

Componentes principales (vista funcional)

- **Gestión de fuentes:** list_sources(), add_source(), remove_source_by_number().
- **Destino y espacio:** autodetect_dest(), config_set_dest(), config_get_dest(), check_space(), check_writable_dest().
- **Respaldo:** backup_incremental(), have_rsync(), run_cmd().
- **Monitor:** monitor_loop(), syslog_line_epoch_or_now(), alert_notify(), send_telegram().
- **Soporte/UX:** demo_lecturas(), diag_tee(), show_last_status(), usage().

4. Flujo principal del menú

```
menu_principal()
 1) menu_config_fuentes()           # Agrega/quita/lista rutas (acepta ~ y $HOME)
 2) select_destination_interactive()# Automático (/mnt/backup|/media/usb) o manual
 3) backup_incremental()            # Respaldo incremental + symlink 'latest'
 4) show_last_status()              # tail de ~/backups/backup.log
 5) monitor_loop(MON_LOG, THR, WIN) # Tail -F y conteo de errores por ventana
 6) demo_lecturas()                 # read -t/-s/-n y select (demo)
 7) diag_tee()                      # tee + exit code de list_sources()
 8) Salir
```

5. Lógica de `backup_incremental()`

1. **Validaciones previas:**
   - `SOURCES_FILE` no vacío y todas las rutas legibles (`check_readable_sources()`).
   - Determinar `DEST` (configurado o autodetectado); verificar escritura y espacio (`check_writable_dest()`, `check_space()`).
2. **Ejecución:**
   - Construye `subdir = DEST/backup_YYYY-MM-DD_HHMMSS`.
   - Si hay `rsync`: usar `-a –update –human-readable –stats` y `–delete` para carpetas (comportamiento tipo espejo). Si no, *fallback* con `cp -a`.
   - Respetar `–dry-run` mediante `run_cmd()` (simula y registra el comando).
3. **Post:** actualizar `DEST/latest` como *symlink* al respaldo creado; registrar INICIO/FIN en el log.

Diagrama de decisiones

```
[check sources] --X--> ERROR (abort)
       |
[resolve DEST + writable + space] --X--> ERROR (abort)
       |
     have_rsync?
     /       \
   yes       no
    |         |
  rsync     cp -a
    |         |
   [update 'latest' + log + OK]
```

6. Lógica del monitor de logs

- **Entrada:** archivo de log (por defecto `~/backup-admin/samples/syslog.sample` o `/var/log/syslog`), umbral N y ventana de M minutos.
- **Detección:** `tail -F` y contador deslizante de marcas que casen `error|fail|critical|segfault|panic`. Cada línea se *timestampea* (si carece de hora, usa `now`).
- **Alerta:** al llegar a N dentro de la ventana M, registrar alerta, simular un reinicio y notificar:
  - *Stub*: imprime "ALERTA (stub) ...".

- **Telegram real**: si existen `TELEGRAM_BOT_TOKEN` y `TELEGRAM_CHAT_ID` y el flag `-force`, envía mensaje vía `send_telegram()`.

7. Validaciones críticas

- **Fuentes:** existencia y permisos de lectura; rutas normalizadas (acepta `~` y `$HOME`).
- **Destino:** escritura y espacio libre (`df` vs. suma `du` aparente de fuentes).
- **Modo prueba:** `-dry-run` conserva toda la lógica sin efectuar cambios, ideal para demostraciones.

8. Interfaz por CLI y registro

- Flags: `-verbose`, `-dry-run`, `-monitor`, `-log`, `-threshold`, `-window`, `-force`.
- **Log**: `/backups/backup.log` centraliza eventos de respaldo/monitor y alertas (con prefijo y hora).

9. Conclusiones

El enfoque *top–down* y la modularidad del script permiten un flujo claro y verificable: configuración → validación → respaldo incremental trazable → monitoreo con umbral y alerta. El uso de `rsync` (con *fallback*), `-dry-run` y un *logging* consistente garantiza seguridad de operación y facilidad de auditoría.

Anexos

- Video demostrativo (YouTube): `https://youtu.be/8Hv-GnnfvDo`
- Repositorio (GitHub): `https://github.com/WillyrexCUE23182/Proyecto-Administrador-ligero-de-respal` `tree/main`