MANUAL TÉCNICO

PRIMER PROYECTO

Ángel Andrés Godoy Valdéz

202113539

Manejo e Implementación de Archivos – Sección "A"

Segundo Semestre 2025

Contenido

Arquitectura del sistema

El simulador ext2 "GoDisk" se compone de un Frontend Angular (UI de consola y modales de reportes) y un Backend en Go (intérprete de comandos y motor EXT2). La comunicación es HTTP local:

- **POST /api/exec**: envía un script de comandos (comandos mkdisk, fdisk, mount, mkfs, mkdir, mkfile, cat, login, rep). El backend no devuelve JSON de reportes aquí, solo la salida textual (stdout/stderr) de la ejecución.
- **GET /api/report**: el frontend pide después los datos estructurados (JSON) de cada reporte: mbr, disk, inode(s), block(s), bm_inode, bm_block, tree, sb, file, ls. Para posteriormente renderizarlos en un modal.

Backend

main.go

- App contiene:
  - reg (mount.Registry) — estado en RAM de discos/particiones montadas.
  - svc (mount.Service) — orquesta Mount() y valida MBR.
  - formatter (ext2.Formatter) — MkfsFull() inicializa EXT2 (SB, BM, inodos, bloques, users.txt).
- NewApp() llama reg.RehydrateFromCatalog() para reconstruir el estado desde:
  - Catálogo (/.godisk/catalog.json) y
  - MBR de cada disco (particiones marcadas como montadas, correlativos, IDs).

internal/commands

- Implementa parsing de flags por comando y llama a servicios:
  - ExecuteMkdisk, ExecuteFdisk (creación disco/partición) → diskio/structs.
  - Mount → mount.Service (y actualiza MBR con status=1, ID, correlativo).
  - Mkfs → ext2.Formatter.MkfsFull.
  - login/logout, mkgrp/rmgrp/mkusr/rmusr/chgrp → lectura/escritura en /users.txt.
  - mkfile/mkdir/cat → internal/ext2 (asignación de inodos/bloques, lectura).
- imprimen mensajes (capturados por ProcessLine).

internal/mount

- Registry: índice en RAM de montajes por disco.
- RehydrateFromCatalog(): lee catalog.All() y MBR de cada ruta válida para reconstruir letras e IDs.
- Service.Mount(): valida MBR y marca part_status=1, guarda part_id y correlative.

internal/ext2

- Mkfs: Formatter.MkfsFull(id) calcula layout, escribe SuperBloque, bitmaps en 0, inodos/blocks, crea / y /users.txt.
- IO: helpers readAt/writeAt, readInodeAt/writeInodeAt, readBlockBytes, bitmaps MarkInode/MarkBlock.
- Users: AppendUsersLine, RewriteUsers, lectura/escritura de /users.txt (CSV).
- FS ops: CreateDir, CreateFile, ReadFile, validación de permisos (básico).

internal/reports

- *Build*/*Generate*: Lógica pura de lectura de disco y armado de modelos JSON:
  - MBR, DISK, INODE, INODES, BLOCK(S), BM_INODE, BM_BLOCK, TREE, SB, FILE, LS.
- Cada GenerateX resuelve un path de salida (JSON/HTML).
- Handlers HTTP devuelven los JSON que consume Angular.

internal/catalog

- Persistencia simple (lista de rutas de .mia) en ~/.godisk/catalog.json.
- Permite rehidratar tras reinicio.

internal/diskio y internal/structs

- Acceso binario a MBR/EBR y structs low-level.

FrontEnd

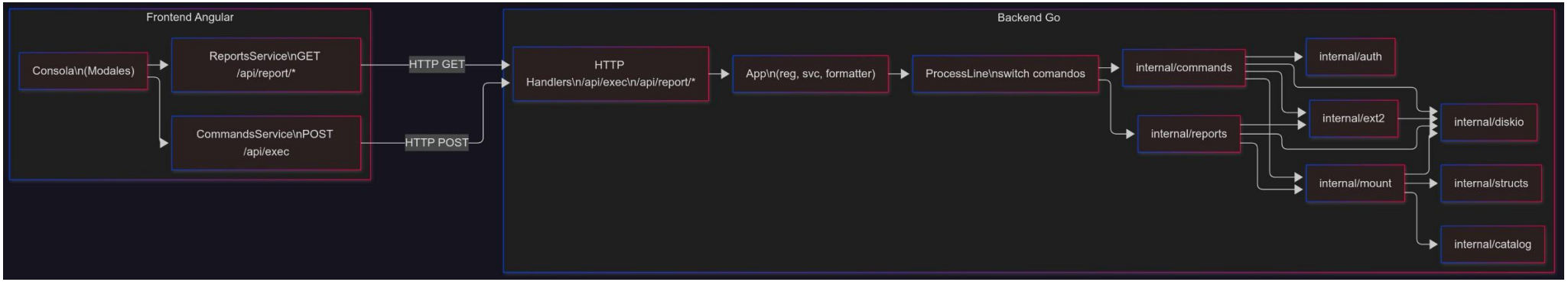
Consola (consola.component):

- Textarea → script multi-línea.
- Click "Ejecutar":
  1. CommandsService.execute(script) → POST /api/exec.
  2. Analiza el script para detectar "rep"
  3. Lanza en paralelo ReportsService (con retry) y abre modales:
     - MBR: tabla de particiones.

- DISK: mosaico proporcional + tabla de segmentos.
- INODE/INODES: detalle y explorador de inodos.
- BLOCK(S): tarjetas por bloque (dir/file/ptr).
- TREE: nodos y aristas, además tabla de todos los inodos y "bloques usados".
- SB: tabla del superbloque.
- FILE: nombre y contenido (txt).
- LS: tabla tipo ls -l (type, perm, owner, group, size, mtime/ctime/atime).

Estados: El componente limpia estados antes de cada ejecución (salida, modales, listas).

Diagrama de Flujo



Estructuras de Datos

MBR (Master Boot Record)

El MBR es el encabezado del disco (offset 0 del .mia). Guarda metadatos del disco y hasta 4 entradas de partición (primarias o una extendida). En el código está modelado en internal/structs/mbr.go (tipo MBR y Partition).

Estructura

- structs.MBR
    - Mbr_tamano (int64): tamaño total del disco en bytes.
    - Mbr_fecha_creacion (int64): timestamp Unix.
    - Mbr_dsk_signature (int64): firma aleatoria.
    - Dsk_fit (byte): BF/FF/WF.
    - Mbr_partitions [4]Partition: 4 slots fijos.
- structs.Partition (slot de partición)
    - Part_status (byte): '1' montada, '0' (u otro) libre/no montada.
    - Part_type (byte): 'p' primaria, 'e' extendida.
    - Part_fit (byte): BF/FF/WF.
    - Part_start (int64): offset absoluto del inicio de la partición dentro del .mia.
    - Part_s (int64): tamaño de la partición en bytes.

EBR (Extended Boot Record)

Si alguna de las 4 entradas del MBR es extendida (Part_type='e'), dentro del rango de esa extendida se encadena una lista de EBRs. Cada EBR describe una partición lógica y un puntero al siguiente EBR.

Estructura

- structs.EBR:
    - Part_status (byte)
    - Part_fit (byte)
    - Part_start (int64)
    - Part_s (int64)
    - Part_next (int64)
    - Part_name [16]byte

Journaling

Ubicada en: \GoDisk\internal\ext3\journal.go

Siendo una bitácora de todas las acciones que se realizan en el sistema de archivos logueado, guarda record de los comandos que modifican archivos.

```go
func readAt(path string, off int64, data any) error {
    f, err := os.Open(path)
    if err != nil {
        return err
    }
    defer f.Close()
    if _, err := f.Seek(off, io.SeekStart); err != nil {
        return err
    }
    return binary.Read(f, binary.LittleEndian, data)
}

func journalRegion(sb ext2.SuperBloque) (off, entries int64) {
    sbSize := xbin.SizeOf[ext2.SuperBloque]()
    jStart := sbSize
    jBytes := sb.SBmInodeStart - jStart
    entrySz := xbin.SizeOf[structs.Journal]()
    if entrySz <= 0 || jBytes <= 0 {
        return 0, 0
    }
    return jStart, jBytes / entrySz
}

func appendJournalEntry(mp *mount.MountedPartition, sb ext2.SuperBloque, entry structs.Journal) error
    jOff, cap := journalRegion(sb)
    if cap <= 0 {
```

Funcionalidad

Cada partición que se formatea con mkfs se inicializa con un SuperBloque. Define cantidades y offsets de todas las áreas del ext2: bitmaps, tabla de inodos y área de bloques.

Marcan ocupación de inodos y bloques

- Bitmap de inodos (SBmInodeStart, tamaño SInodesCount bytes).
- Bitmap de bloques (SBmBlockStart, tamaño SBlocksCount bytes).

Cada inodo describe un archivo o directorio: dueño, permisos, tamaño, punteros a bloques, tiempos, etc.

Modulos:

- internal/diskio: lectura/escritura binaria de MBR/EBR.
- internal/mount:
    - Service.Mount: valida MBR, marca Part_status='1', setea Part_correlative e id, y guarda en RAM el MountedPartition (start/size).
    - Registry.RehydrateFromCatalog: lee catalog.json y MBR de cada ruta para reconstruir letras/IDs/correlativos y las entradas montadas (solo si Part_status='1').
- internal/ext2:
    - Formatter.MkfsFull: calcula layout, escribe SB, bitmaps en 0, tabla de inodos, crea inode(users.txt) y sus bloques, marca bitmaps.

    - CreateDir, CreateFile, ReadFile: efectúan la lógica de asignación de inodos/bloques y entradas de directorio.
    - users: lectura/escritura de users.txt.
- internal/reports:
    - mbr/disk: inspeccionan MBR + EBRs.
    - sb: muestra el SuperBloque (todos los campos).
    - inode/inodes/tree: leen inodos, resuelven punteros (directos/indirectos) y entradas de directorio.
    - block/bm_inode/bm_block: muestran bloques usados y bitmaps.
    - file/ls: despliegan contenido/propiedades.

Comandos

Mkdisk

- Crea archivo .mia con tamaño fijo (relleno).
- Escribe MBR en offset=0: mbr_tamano, fecha, signature, dsk_fit, 4 Partition vacías.
- Agrega la ruta al catálogo.

GoDisk\internal\commands\mkdisk.go:

```go
// ExecuteMkdisk contiene la lógica principal para crear un disco virtual.
// Esta función es exportada (empieza con mayúscula) para que pueda ser llamada desde otros paquetes
func ExecuteMkdisk(size int, unit string, fit string, path string) {

    // Declara variable para almacenar el tamaño final en bytes
    // Se usa int64 para soportar discos grandes (hasta 9 exabytes teóricamente)
    var diskSize int64

    // Validación completa del parámetro unit (unidad de medida)
    // Convierte a mayúsculas para hacer comparación insensible a mayúsculas/minúsculas
    unit = strings.ToUpper(unit)

    if unit == "K" {
        // Si la unidad es K (kilobytes), multiplica por 1024
        // 1 KB = 1024 bytes (sistema binario, no decimal)
        diskSize = int64(size) * 1024
    } else if unit == "M" || unit == "" {
        // Si la unidad es M (megabytes) o está vacía (valor por defecto)
        // 1 MB = 1024 * 1024 = 1,048,576 bytes
        diskSize = int64(size) * 1024 * 1024
    } else {
        // Si la unidad no es válida, mostrar error y terminar función
        fmt.Printf("Error: valor '%s' no válido para -unit. Use K o M.\n", unit)
        return // Termina la ejecución de la función
    }

    // Validación adicional: el tamaño debe ser positivo
    if diskSize <= 0 {
        fmt.Println("Error: el parámetro -size debe ser mayor a cero.")
        return
    }

    // Variable para almacenar el byte que representa el tipo de ajuste
    var fitByte byte

    // Convierte a mayúsculas para comparación insensible a mayúsculas/minúsculas
    fit = strings.ToUpper(fit)
```

Uso: mkdisk -size=<entero> [-unit=k|m] [-fit=bf|ff|wf] -path="<ruta/disco.mia>"

Rmdisk

- Elimina el archivo .mia si existe (sin interacción).
- Quita la ruta del catálogo y fuerza rehidratación del Registry (descarta montajes en RAM de ese disco).
- Si el archivo no existe: no falla la ejecución; simplemente limpia el catálogo (comportamiento tolerante).

GoDisk\internal\commands\rmdisk.go:

```go
package commands

import (
    "fmt"
    "os"
    "path/filepath"
    "strings"
)

// ExecuteRmdisk elimina el archivo .mia de forma no interactiva.
//   - Si el archivo no existe, lo toma como éxito (idempotente).
//   - Nunca bloquea ni termina el proceso; solo imprime y retorna un bool
//     que indica si se debe remover del catálogo.
func ExecuteRmdisk(path string) bool {
    path = strings.TrimSpace(path)
    if path == "" {
        fmt.Println("rmdisk: -path requerido")
        return false
    }
    ap := filepath.Clean(path)

    if err := os.Remove(ap); err != nil {
        if os.IsNotExist(err) {
            fmt.Println("rmdisk: el archivo ya no existe (idempotente)")
            return true
        }
        fmt.Printf("rmdisk: no se pudo eliminar %q: %v\n", ap, err)
        return false
    }

    fmt.Println("rmdisk: eliminado", ap)
    return true
}
```

Uso: rmdisk -path="<ruta/disco.mia>"

Fdisk

- MBR: ocupa uno de los 4 slots si -type=p|e, calculando part_start y part_s según fit.
- EBR: crea un EBR inicial o encadena (part_next) para -type=l dentro de la región extendida.
- No toca EXT2 (eso lo hace mkfs).

GoDisk\internal\commands\fdisk.go:

```go
// ExecuteFdisk es el punto de entrada principal para el comando fdisk.
// Decide qué tipo de partición crear y llama a la función correspondiente.
func ExecuteFdisk(path, name, unit, typeStr, fit string, size int64) {
    // 1. Abrir el archivo del disco en modo lectura/escritura
    file, err := os.OpenFile(path, os.O_RDWR, 0644)
    if err != nil {
        if os.IsNotExist(err) {
            fmt.Printf("Error: el disco en la ruta '%s' no existe.\n", path)
        } else {
            fmt.Printf("Error al abrir el disco: %v\n", err)
        }
        return
    }
    defer file.Close()

    // 2. Leer el MBR existente del disco
    var mbr structs.MBR
    file.Seek(0, 0)
    err = binary.Read(file, binary.LittleEndian, &mbr)
    if err != nil {
        fmt.Printf("Error al leer el MBR del disco: %v\n", err)
        return
    }

    // 3. Calcular el tamaño de la nueva partición en bytes
    var partitionSize int64
    switch strings.ToLower(unit) {
    case "b":
        partitionSize = size
    case "m":
        partitionSize = size * 1024 * 1024
    default: // "k" es el default
        partitionSize = size * 1024
    }

    // 4. Llamar a la función correcta según el tipo de partición
    switch strings.ToLower(typeStr) {
    case "p":
```

Uso: fdisk -size=<entero> -path="<disco.mia>" -name="<nombre>" [-unit=b|k|m] [-type=p|e|l] [-fit=bf|ff|wf]

Mount

- Valida MBR y que exista la partición.
- Asigna letra de disco (A..Z) y correlativo siguiente.
- Genera ID con prefijo "39".
- Marca en MBR: part_status='1', part_correlative=<n>, copia part_id.
- Registra en Registry en RAM (para mounted, mkfs, etc.).
- Persistente: al reiniciar, RehydrateFromCatalog() lee MBR y reconstruye montajes con status='1'.

GoDisk\internal\mount\listed.go:

```go
import (
    "bytes"
    "fmt"
    "sort"
    "strconv"
    "strings"
)

type MPView struct {
    ID       string `json:"id"`
    DiskPath string `json:"disk_path"`
    Letter   string `json:"letter"`
    Number   int    `json:"number"`
    PartName string `json:"part_name"`
    Start    int64  `json:"start"`
    Size     int64  `json:"size"`
}

// MountedPlain devuelve una línea con los IDs montados separados por coma,
func (r *Registry) MountedPlain() (string, error) {
    views := r.mountedSnapshot()
    if len(views) == 0 {
        return "", ErrNotMounted
    }
    ids := make([]string, len(views))
    for i, v := range views {
        ids[i] = v.ID
    }
    return strings.Join(ids, ", "), nil
}

// MountedJSON devuelve una porción de MPView lista para serializar como JSON.
func (r *Registry) MountedJSON() ([]MPView, error) {
    views := r.mountedSnapshot()
    if len(views) == 0 {
        return nil, ErrNotMounted
    }
    return views, nil
}
```

Uso: mount -path="<disco.mia>" -name="<nombre_particion_primaria>"

Mounted

- Lee el Registry en RAM (previamente rehidratado del catálogo + MBR).
- No modifica estructuras.

GoDisk\internal\commands\cmd_mounted.go:

```go
//   - -json:     imprime JSON
func CmdMounted(reg *mount.Registry, argv []string) int {
    flags := parseFlags(argv)

    // JSON
    if flags["-json"] {
        views, err := reg.MountedJSON()
        if err != nil {
            fmt.Println("(sin particiones montadas)")
            return 0
        }
        b, _ := json.MarshalIndent(views, "", "  ")
        fmt.Println(string(b))
        return 0
    }

    // Tabla
    if flags["-table"] {
        table, err := reg.MountedTable()
        if err != nil {
            fmt.Println("(sin particiones montadas)")
            return 0
        }
        fmt.Print(table)
        return 0
    }

    // Plano
    line, err := reg.MountedPlain()
    if err != nil {
        fmt.Println("(sin particiones montadas)")
        return 0
    }
    fmt.Println(line)
    return 0
}
```

Uso: mounted

Mkfs

- Escribe SuperBloque (cantidades y offsets).
- Inicializa bitmaps (inodos/bloques) en cero.
- Crea inodo raíz / (tipo dir) con su bloque de directorio.
- Crea /users.txt (tipo file), lo inicializa con:

GoDisk\internal\ext2\mkfs_service.go:

```go
package ext2

import (
    "fmt"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

type Formatter struct {
    reg *mount.Registry
}

func NewFormatter(reg *mount.Registry) *Formatter { return &Formatter{reg: reg} }

// MkfsFull realiza el formateo completo EXT2 en la partición identificada por id.
func (f *Formatter) MkfsFull(id string) error {
    mp, ok := f.reg.GetByID(id)
    if !ok {
        return fmt.Errorf("mkfs: id %s no está montado", id)
    }

    partStart := mp.Start
    partSize := mp.Size

    // Cálculo de n y layout (SB + offsets) — enunciado EXT2
    _, sb, err := ComputeLayout(partSize)
    if err != nil {
        return err
    }

    // Escribir SuperBloque al inicio de la partición
    if err := writeAt(mp.DiskPath, partStart, sb); err != nil {
        return fmt.Errorf("mkfs: error escribiendo superbloque: %w", err)
    }

    //  Inicializar bitmaps
    bmIn, bmBl := NewBitmaps(sb.SInodesCount, sb.SBlocksCount)

    // Reservar inodos/bloques para raíz y users.txt
    // inodo 0 (carpeta) con bloque 0; inodo 1 (archivo) con bloque 1
    MarkInode(bmIn, 0, true)
    MarkBlock(bmBl, 0, true)
```

Uso: mkfs -id=<ID_montaje> [-type=full]

Login

- Abre el EXT2, lee y parsea /users.txt.
- Valida usuario/contraseña y setea sesión en internal/auth (en memoria de la app).
- No toca bitmaps ni MBR.

GoDisk\internal\auth\session.go:

```go
package auth

import (
    "errors"
    "fmt"
    "strings"
    "sync"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

type Session struct {
    ID     string // ID de partición montada
    User   string
    Group  string
    UID    int
    GID    int
    IsRoot bool
}

var (
    mu      sync.RWMutex
    current *Session
)

func Current() (*Session, bool) {
    mu.RLock()
    defer mu.RUnlock()
    if current == nil {
        return nil, false
    }
    c := *current
    return &c, true
}

func Logout() {
    mu.Lock()
    defer mu.Unlock()
    current = nil
}
```

Uso: login -id=<ID> -user=<usuario> -pass=<password>

Logout

- Limpia la sesión activa (en internal/auth).
- No altera disco.

```go
package auth

import (
    "errors"
    "fmt"
    "strings"
    "sync"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

type Session struct {
    ID     string // ID de partición montada
    User   string
    Group  string
    UID    int
    GID    int
    IsRoot bool
}

var (
    mu      sync.RWMutex
    current *Session
)

func Current() (*Session, bool) {
    mu.RLock()
    defer mu.RUnlock()
    if current == nil {
        return nil, false
    }
    c := *current
    return &c, true
}

func Logout() {
    mu.Lock()
    defer mu.Unlock()
    current = nil
}
```

Uso: logout

Mkgrp

- Verifica sesión válida (root).
- Append a /users.txt: id,G,<grupo> con correlativo siguiente.
- Reescribe el archivo (si corresponde) y actualiza tamaño de archivo según contenido.
- No cambia bitmaps (es el mismo inodo/bloques; solo reescritura de datos).

GoDisk\internal\usersvc\mkgrp.go:

```go
package usersvc

import (
    "errors"
    "fmt"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Mkgrp crea un grupo nuevo en /users.txt.
func Mkgrp(reg *mount.Registry, name string) error {
    name = strings.TrimSpace(name)
    if name == "" || strings.ContainsAny(name, " \t\r\n") || strings.ContainsRune(name, ',') {
        return errors.New("mkgrp: nombre inválido (no puede estar vacío ni contener espacios o comas)")
    }

    // Sesión actual
    s, ok := auth.Current()
    if !ok {
        return errors.New("mkgrp: no hay sesión activa; usa login")
    }
    if !s.IsRoot {
        return errors.New("mkgrp: operación permitida solo para root")
    }

    // Leer contenido actual
    txt, err := ext2.ReadUsersText(reg, s.ID)
    if err != nil {
        return fmt.Errorf("mkgrp: %w", err)
    }

    // Parsear grupos: "gid, G, nombre"
    maxGID := 0
    for _, line := range splitLines(txt) {
        line = strings.TrimSpace(line)
        if line == "" || strings.HasPrefix(line, "#") {
            continue
        }
        parts := splitCSV(line)
```

Uso: mkgrp -name=<grupo>

Rmgrp

- Marca como eliminado (convención: cambia id a 0 o línea desactivada) dentro de /users.txt mediante RewriteUsers.
- No libera inodos/bloques, solo reescribe contenido del archivo.

GoDisk\internal\usersvc\rmgrp.go

```go
package usersvc

import (
    "errors"
    "fmt"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Rmgrp elimina lógicamente un grupo en /users.txt marcándolo con gid=0.
// Reglas:
//   - Requiere sesión activa y root.
//   - El grupo debe existir y estar ACTIVO (gid != 0).
//   - No cambia usuarios; solo marca el grupo como eliminado lógico (como exige el enunciado).
func Rmgrp(reg *mount.Registry, name string) error {
    name = strings.TrimSpace(name)
    if name == "" || strings.ContainsRune(name, ',') {
        return errors.New("rmgrp: nombre inválido (no vacío ni con comas)")
    }

    s, ok := auth.Current()
    if !ok {
        return errors.New("rmgrp: no hay sesión activa; usa login")
    }
    if !s.IsRoot {
        return errors.New("rmgrp: operación permitida solo para root")
    }

    // Lee /users.txt
    txt, err := ext2.ReadUsersText(reg, s.ID)
    if err != nil {
        return fmt.Errorf("rmgrp: %w", err)
    }

    // Recorrer y marcar línea del grupo como "0, G, <name>"
    lines := splitLines(txt)
    found := false
    alreadyDeleted := false
```

Uso: rmgrp -name=<grupo>

Mkusr

- Valida que el grupo exista y usuario no exista.
- Agrega línea a /users.txt: id,U,<grp>,<user>,<pass>.
- Reescritura de contenido del archivo.

GoDisk\internal\usersvc\mkusr.go:

```go
package usersvc

import (
    "errors"
    "fmt"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Mkusr crea un usuario nuevo en /users.txt.
// Reglas:
//   - Requiere sesión activa y que sea root.
//   - -usr, -pass, -grp obligatorios, sin espacios ni comas.
//   - El grupo debe existir y estar ACTIVO (gid != 0).
//   - No permite duplicar usuario ACTIVO (uid != 0).
//   - El nuevo UID = max(UID activos) + 1.
func Mkusr(reg *mount.Registry, usr, pass, grp string) error {
    usr = strings.TrimSpace(usr)
    pass = strings.TrimSpace(pass)
    grp = strings.TrimSpace(grp)

    if usr == "" || pass == "" || grp == "" {
        return errors.New("mkusr: faltan -usr, -pass o -grp")
    }
    if invalidToken(usr) || invalidToken(pass) || invalidToken(grp) {
        return errors.New("mkusr: usr/pass/grp no deben contener espacios ni comas")
    }

    s, ok := auth.Current()
    if !ok {
        return errors.New("mkusr: no hay sesión activa; usa login")
    }
    if !s.IsRoot {
        return errors.New("mkusr: operación permitida solo para root")
    }

    txt, err := ext2.ReadUsersText(reg, s.ID)
    if err != nil {
        return fmt.Errorf("mkusr: %w", err)
```

Uso: mkusr -user=<usuario> -pass=<pass> -grp=<grupo>

Rmusr

- Marca como eliminado al usuario en /users.txt (misma lógica que rmgrp).
- Reescritura del archivo.

GoDisk\internal\usersvc\rmusr.go:

```go
package usersvc

import (
    "errors"
    "fmt"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Rmusr elimina lógicamente un usuario en /users.txt marcándolo con uid=0.
func Rmusr(reg *mount.Registry, usr string) error {
    usr = strings.TrimSpace(usr)
    if usr == "" || invalidToken(usr) {
        return errors.New("rmusr: -usr inválido (no vacío, sin espacios ni comas)")
    }
    if strings.EqualFold(usr, "root") {
        return errors.New("rmusr: no se permite eliminar al usuario root")
    }

    s, ok := auth.Current()
    if !ok {
        return errors.New("rmusr: no hay sesión activa; usa login")
    }
    if !s.IsRoot {
        return errors.New("rmusr: operación permitida solo para root")
    }

    txt, err := ext2.ReadUsersText(reg, s.ID)
    if err != nil {
        return fmt.Errorf("rmusr: %w", err)
    }

    // Marcar usuario con uid=0 preservando grupo y contraseña
    lines := splitLines(txt)
    found := false
    alreadyDeleted := false

    for i, raw := range lines {
        line := strings.TrimSpace(raw)
```

Uso: rmusr -user=<usuario>

Chgrp

- Reescribe la línea del usuario en /users.txt cambiando el grupo.
- Mantiene inodo/bloques; solo cambia el contenido.

GoDisk\internal\usersvc\chgrp.go:

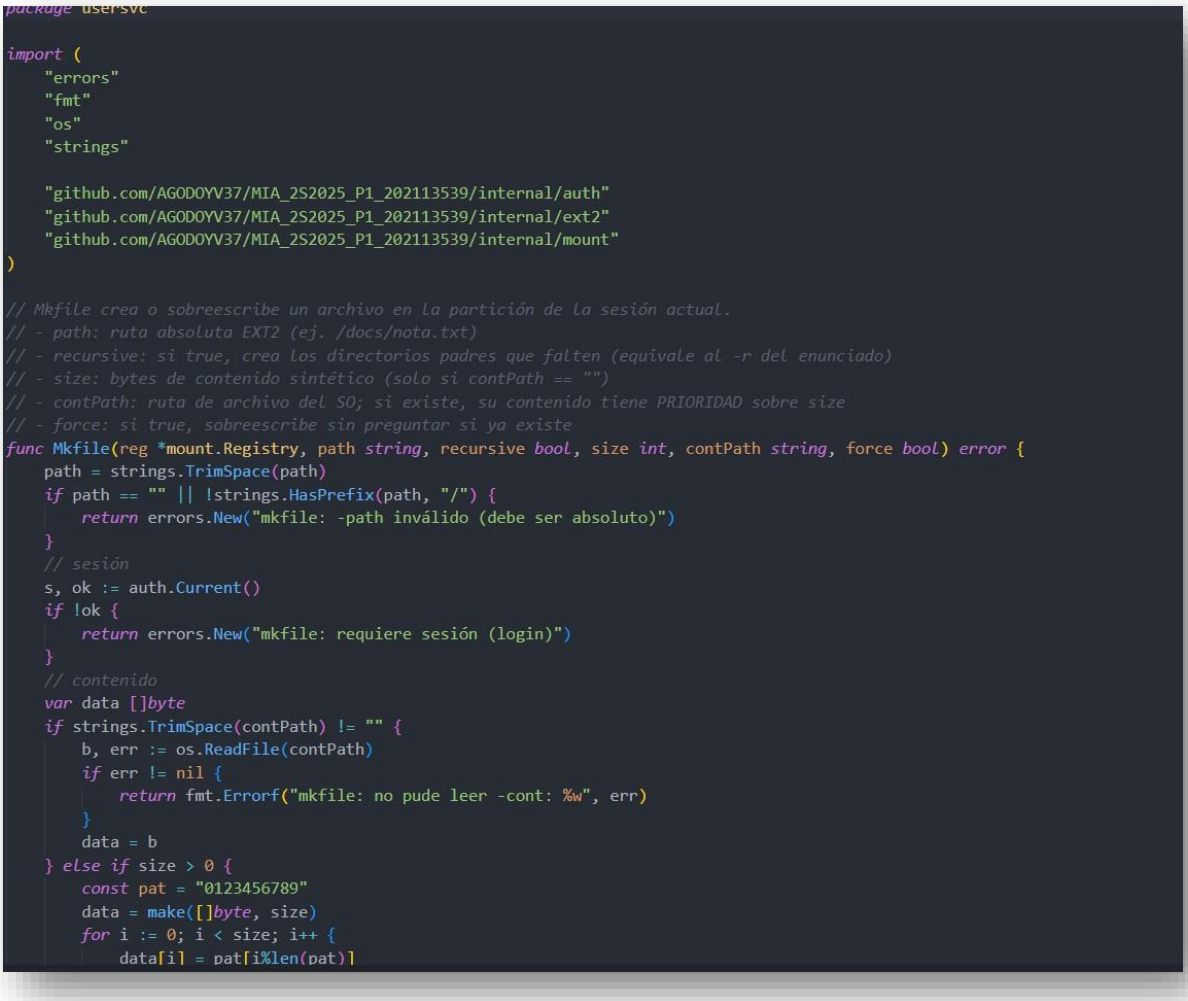
```go
import (
    "errors"
    "fmt"
    "strings"
    "unicode"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Chgrp cambia el grupo de un usuario en /users.txt (borrado lógico style).
func Chgrp(reg *mount.Registry, usr, newGrp string) error {
    usr = strings.TrimSpace(usr)
    newGrp = strings.TrimSpace(newGrp)

    if usr == "" || newGrp == "" {
        return errors.New("chgrp: faltan -usr o -grp")
    }
    // Validación inline para evitar helpers duplicados
    if strings.ContainsRune(usr, ',') || strings.ContainsRune(newGrp, ',') {
        return errors.New("chgrp: usr/grp no deben contener comas")
    }
    for _, r := range usr {
        if unicode.IsSpace(r) {
            return errors.New("chgrp: usr no debe contener espacios")
        }
    }
    for _, r := range newGrp {
        if unicode.IsSpace(r) {
            return errors.New("chgrp: grp no debe contener espacios")
        }
    }
    if strings.EqualFold(usr, "root") {
        return errors.New("chgrp: no se permite cambiar el grupo del usuario root")
    }

    s, ok := auth.Current()
    if !ok {
        return errors.New("chgrp: no hay sesión activa; usa login")
```

Uso: chgrp -user=<usuario> -grp=<nuevo_grupo>

Mkfile

- Resuelve directorios, crea inodo de archivo si no existe y bloques de datos necesarios.
- Escribe contenido y actualiza i_size.
- Actualiza bitmap de inodos/bloques si se asignan nuevos.
- Agrega entrada en el bloque de directorio padre.

GoDisk\internal\usersvc\mkfile.go:



```go
package usersvc

import (
    "errors"
    "fmt"
    "os"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Mkfile crea o sobreescribe un archivo en la partición de la sesión actual.
// - path: ruta absoluta EXT2 (ej. /docs/nota.txt)
// - recursive: si true, crea los directorios padres que falten (equivale al -r del enunciado)
// - size: bytes de contenido sintético (solo si contPath == "")
// - contPath: ruta de archivo del SO; si existe, su contenido tiene PRIORIDAD sobre size
// - force: si true, sobreescribe sin preguntar si ya existe
func Mkfile(reg *mount.Registry, path string, recursive bool, size int, contPath string, force bool) error {
    path = strings.TrimSpace(path)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("mkfile: -path inválido (debe ser absoluto)")
    }
    // sesión
    s, ok := auth.Current()
    if !ok {
        return errors.New("mkfile: requiere sesión (login)")
    }
    // contenido
    var data []byte
    if strings.TrimSpace(contPath) != "" {
        b, err := os.ReadFile(contPath)
        if err != nil {
            return fmt.Errorf("mkfile: no pude leer -cont: %w", err)
        }
        data = b
    } else if size > 0 {
        const pat = "0123456789"
        data = make([]byte, size)
        for i := 0; i < size; i++ {
            data[i] = pat[i%len(pat)]
```

Uso: mkfile -path="/abs/ruta/archivo.txt" [-p] [-size=<bytes>] [-cont="/abs/ruta/origen.txt" | -cont="texto literal"]

Mkdir

- Crea inodo directorio y su bloque de directorio (con . y ..).
- Enlaza entrada en directorio padre.
- Marca bitmaps de inodo/bloque si son nuevos.

GoDisk\internal\usersvc\mkdir.go:

```go
package usersvc

import (
    "errors"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Mkdir crea un directorio en la partición de la sesión actual.
// - path: ruta absoluta EXT2 (/a/b/c)
// - p: si true, crea padres faltantes (equivale a -p)
// Requiere sesión activa (no necesariamente root).
func Mkdir(reg *mount.Registry, path string, p bool) error {
    path = strings.TrimSpace(path)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("mkdir: -path inválido (debe ser absoluto)")
    }
    s, ok := auth.Current()
    if !ok {
        return errors.New("mkdir: requiere sesión (login)")
    }
    return ext2.MakeDir(reg, s.ID, path, p, s.UID, s.GID)
}
```

Uso: mkdir -path="/abs/ruta/dir" [-p]

Cat

- Resuelve inodo del archivo, lee sus bloques de datos y los imprime concatenados (sin modificar disco).

GoDisk\internal\usersvc\cat.go:

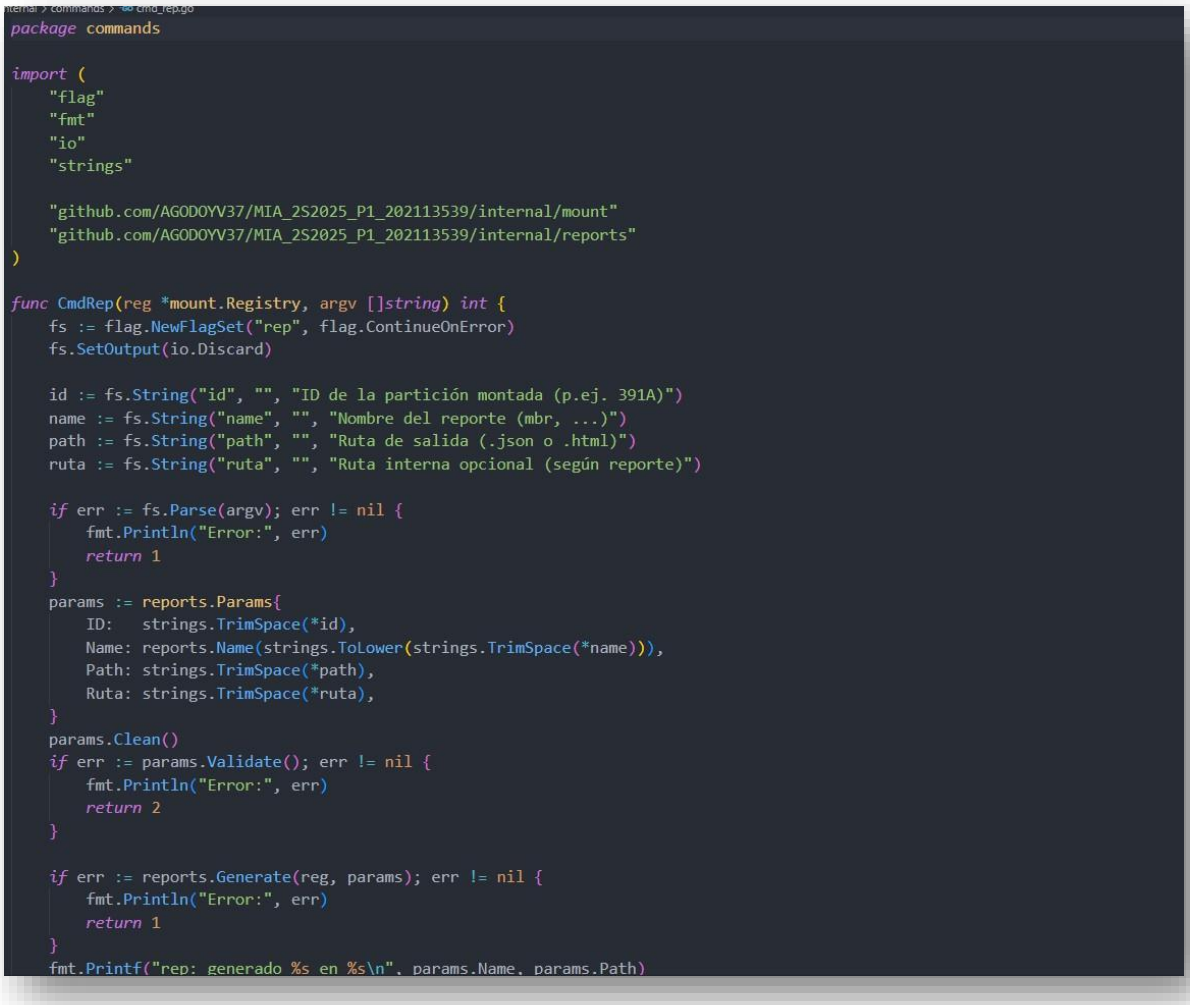
```go
package usersvc

import (
    "errors"
    "fmt"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/auth"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/ext2"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
)

// Cat concatena el contenido de varios archivos (en orden).
func Cat(reg *mount.Registry, files []string) (string, error) {
    if len(files) == 0 {
        return "", errors.New("cat: especifica al menos un -fileN")
    }
    s, ok := auth.Current()
    if !ok {
        return "", errors.New("cat: requiere sesión (login)")
    }

    var b strings.Builder
    for i, path := range files {
        path = strings.TrimSpace(path)
        if path == "" || !strings.HasPrefix(path, "/") {
            return "", fmt.Errorf("cat: ruta inválida: %q", path)
        }
        ino, data, err := ext2.ReadFileByPath(reg, s.ID, path)
        if err != nil {
            return "", err
        }
        if !canRead(ino, s) {
            return "", fmt.Errorf("cat: permiso denegado para %s", path)
        }
        if i > 0 {
            b.WriteByte('\n')
        }
        b.Write(data)
    }
    return b.String(), nil
}
```

Uso: cat -file1=/abs/ruta/a.txt [-file2=...] [-file3=...]

Rep

Con varios inputs, no tiene efectos en el backend, simplemente muestra en un modal la información según el reporte.

GoDisk\internal\commands\cmd_rep.go



```go
package commands

import (
    "flag"
    "fmt"
    "io"
    "strings"

    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/mount"
    "github.com/AGODOYV37/MIA_2S2025_P1_202113539/internal/reports"
)

func CmdRep(reg *mount.Registry, argv []string) int {
    fs := flag.NewFlagSet("rep", flag.ContinueOnError)
    fs.SetOutput(io.Discard)

    id := fs.String("id", "", "ID de la partición montada (p.ej. 391A)")
    name := fs.String("name", "", "Nombre del reporte (mbr, ...)")
    path := fs.String("path", "", "Ruta de salida (.json o .html)")
    ruta := fs.String("ruta", "", "Ruta interna opcional (según reporte)")

    if err := fs.Parse(argv); err != nil {
        fmt.Println("Error:", err)
        return 1
    }
    params := reports.Params{
        ID:   strings.TrimSpace(*id),
        Name: reports.Name(strings.ToLower(strings.TrimSpace(*name))),
        Path: strings.TrimSpace(*path),
        Ruta: strings.TrimSpace(*ruta),
    }
    params.Clean()
    if err := params.Validate(); err != nil {
        fmt.Println("Error:", err)
        return 2
    }

    if err := reports.Generate(reg, params); err != nil {
        fmt.Println("Error:", err)
        return 1
    }
    fmt.Printf("rep: generado %s en %s\n", params.Name, params.Path)
```

Uso: rep -id=<ID> -name=<mbr|disk|inode|inodes|block|bm_inode|bm_block|tree|sb|file|ls> -ruta="/ruta de archivo.txt"

Unmount

Desmonta una partición del sistema.

GoDisk\internal\mount\service.go

```go
// Unmount por ID
func (s *Service) UnmountByID(id string) error {
    mp, ok := s.reg.RemoveByID(id)
    if !ok {
        return ErrIDNotFound
    }
    // limpia Part_id y Part_correlative en el MBR
    if err := clearMBRMountMeta(mp.DiskPath, mp.PartName); err != nil {
        return Wrap(ErrMBRWrite, "unmount: %v", err)
    }
    // si ya no quedan particiones montadas de ese disco, libera la letra
    _ = s.reg.PurgeDiskIfEmpty(mp.DiskPath)
    return nil
}
```

Uso: unmount -id=<ID>

Remove

Este comando permitr eliminar un archivo o carpeta y todo su contenido, si el usuario que actualmente está logueado tiene acceso al permiso de escritura sobre el archivo y en el caso de carpetas, elimina todos los archivos o subcarpetas en los que el usuario tenga permiso de escritura.

GoDisk\internal\usersvc\remove.go:

```go
func Remove(reg *mount.Registry, path string) error {
    path = strings.TrimSpace(path)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("remove: -path inválido (debe ser absoluto)")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("remove: requiere sesión (login)")
    }

    if err := ext2.Remove(reg, s.ID, path, s.UID, s.GID); err != nil {
        return err
    }

    _ = ext3.AppendJournalIfExt3(reg, s.ID, "REMOVE", path, "")

    return nil
}
```

remove -path=<path>

Edit

Este comando permite editar el contenido de un archivo para asignarle otro contenido.

"GoDisk\internal\usersvc\edit.go":

```go
func Edit(reg *mount.Registry, path string, cont string) error {
    path = strings.TrimSpace(path)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("edit: -path inválido (debe ser absoluto)")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("edit: requiere sesión (login)")
    }

    data, err := resolveEditContent(cont)
    if err != nil {
        return err
    }

    // Editar el archivo (requiere rw o root, validado en ext2.EditFile)
    if err := ext2.EditFile(reg, s.ID, path, data, s.UID, s.GID, s.IsRoot); err != nil {
        return err
    }

    // Journaling
      = ext3.AppendJournalTfFxt3(reg, s.ID, "EDIT", path, string(data))
```

Uso: edit -path=<path> -contenido=<path>

Rename

Este comando permite cambiar el nombre de un archivo o carpeta, si el usuario actualmente logueado tiene permiso de escritura sobre el archivo o carpeta.

"GoDisk\internal\usersvc\rename.go":

```go
func Rename(reg *mount.Registry, path, newName string) error {
    path = strings.TrimSpace(path)
    newName = strings.TrimSpace(newName)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("rename: -path inválido (debe ser absoluto)")
    }
    if newName == "" {
        return errors.New("rename: -name requerido")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("rename: requiere sesión (login)")
    }
```

uso: rename -path=<path> -name=<path>

Copy

Este comando permite realizar una copia del archivo o carpeta y todo su contenido hacia otro destino. Tendrá los siguientes parámetros.

"GoDisk\internal\usersvc\copy.go":

```go
func Rename(reg *mount.Registry, path, newName string) error {
    path = strings.TrimSpace(path)
    newName = strings.TrimSpace(newName)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("rename: -path inválido (debe ser absoluto)")
    }
    if newName == "" {
        return errors.New("rename: -name requerido")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("rename: requiere sesión (login)")
    }

    if err := ext2.RenameNode(reg, s.ID, path, newName, s.UID, s.GID, s.IsRoot); err != nil {
        return err
    }
    _ = ext3.AppendJournalIfExt3(reg, s.ID, "RENAME", path, "name="+newName)
    return nil

}
```

Uso: copy -path=<path> -destino=<path>

Move

Este comando mueve un archivo o carpeta y todo su contenido hacia otro destino. Si el origen y destino están dentro de la misma partición, solo cambiará las referencias, para que ya no tenga el padre origen sino, el padre destino, y que los padres de la carpeta o archivo ya no tengan como hijo a la carpeta o archivo que se movió

"GoDisk\internal\usersvc\move.go":

```go
func Move(reg *mount.Registry, src, dst string) error {
    src = strings.TrimSpace(src)
    dst = strings.TrimSpace(dst)
    if src == "" || !strings.HasPrefix(src, "/") {
        return errors.New("move: -path inválido (debe ser absoluto)")
    }
    if dst == "" || !strings.HasPrefix(dst, "/") {
        return errors.New("move: -destino inválido (debe ser absoluto)")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("move: requiere sesión (login)")
    }

    if err := ext2.MoveNode(reg, s.ID, src, dst, s.UID, s.GID, s.IsRoot); err != nil {
        return err
    }

    _ = ext3.AppendJournalIfExt3(reg, s.ID, "MOVE", src, "dest="+dst)

    return nil
}
```

Uso: move -path=<path> destino=<path>

Find

Este comando permite realizar una búsqueda por el nombre del archivo o carpeta.

"GoDisk\internal\usersvc\find.go":

```go
func Find(reg *mount.Registry, startPath, namePattern string) ([]string, error) {
    startPath = strings.TrimSpace(startPath)
    if startPath == "" || !strings.HasPrefix(startPath, "/") {
        return nil, errors.New("find: -path inválido (debe ser absoluto)")
    }
    if strings.TrimSpace(namePattern) == "" {
        namePattern = "*"
    }

    s, err := auth.Require()
    if err != nil {
        return nil, errors.New("find: requiere sesión (login)")
    }

    return ext2.Find(reg, s.ID, startPath, namePattern, s.UID, s.GID, s.IsRoot)
}
```

Uso: find -path = "/" -name=*

Chown

Cambia el propietario de uno o varios archivos o carpetas. Lo podrá utilizar el usuario root en todos los archivos o carpetas y también lo podrán utilizar otros usuarios, pero solo sobre sus propios archivos.

```go
func Chown(reg *mount.Registry, path, newUser string, recursive bool) error {
    path = strings.TrimSpace(path)
    newUser = strings.TrimSpace(newUser)
    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("chown: -path inválido (debe ser absoluto)")
    }
    if newUser == "" {
        return errors.New("chown: -usuario requerido")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("chown: requiere sesión (login)")
    }

    if err := ext2.Chown(reg, s.ID, path, newUser, recursive, s.UID, s.GID, s.IsRoot); err != n
        return err
    }

    _ = ext3.AppendJournalIfExt3(reg, s.ID, "CHOWN", path, fmt.Sprintf("usuario=%s recursive=%t

    return nil
}
```

Uso: Chown -path=<path> -usuario=<user>

Chmod

Este comando elimina un usuario en la partición.

```go
func Chmod(reg *mount.Registry, path, ugo string, recursive bool) error {
    path = strings.TrimSpace(path)
    ugo = strings.TrimSpace(ugo)

    if path == "" || !strings.HasPrefix(path, "/") {
        return errors.New("chmod: -path inválido (debe ser absoluto)")
    }
    if len(ugo) != 3 {
        return errors.New("chmod: -ugo debe tener exactamente 3 dígitos (ej. 764)")
    }

    s, err := auth.Require()
    if err != nil {
        return errors.New("chmod: requiere sesión (login)")
    }
    // Solo root puede ejecutar chmod
    if !s.IsRoot {
        return errors.New("chmod: operación permitida solo para root")
    }

    perms, err := ext2.ParseUGO(ugo)
    if err != nil {
        return fmt.Errorf("chmod: %w", err)
    }
```

Uso: chmod -path=<path> -r -ugo=[0-7][0-7][0-7]