Comandos Útiles - Gestión de Cache y Deploy

🚀 Uso Rápido

Verificación Pre-Deploy (Ejecutar SIEMPRE antes de push)

```
cd /home/ubuntu/escalafin_mvp
./scripts/pre-deploy-verification.sh
```

Diagnóstico de Problemas de Cache

```
cd /home/ubuntu/escalafin_mvp
./scripts/cache-diagnostics.sh
```

📝 Comandos Git Útiles

Verificar estado del repositorio

```
git status
git log -1  # Ver último commit
git diff    # Ver cambios sin commitear
```

Sincronizar con GitHub

```
# Ver diferencias con origin
git fetch origin
git log HEAD..origin/main --oneline

# Ver últimos 5 commits
git log -5 --oneline --graph

# Ver qué archivos cambiaron en el último commit
git show --name-only
```

Hacer commit y push correctamente

```
# 1. Verificar cambios
git status

# 2. Agregar archivos
git add .

# 3. Ver qué se va a commitear
git diff --staged

# 4. Commit con mensaje descriptivo
git commit -m "fix: corregir problema de cache en Dockerfile"

# 5. Push a GitHub
git push origin main

# 6. Verificar que llegó
git log origin/main -1
```

🐳 Comandos Docker Locales

Ver estado de contenedores

```
docker ps              # Contenedores corriendo
docker ps -a           # Todos los contenedores
docker images          # Ver imágenes
```

Limpiar Docker local

```
# Limpiar contenedores detenidos
docker container prune -f

# Limpiar imágenes sin usar
docker image prune -a -f

# Limpiar TODO el cache de build
docker builder prune -a --force

# Limpiar volúmenes sin usar
docker volume prune -f

# Limpiar TODA la cache (⚠️ CUIDADO)
docker system prune -a --volumes -f
```

Build local para testing

```
# Build sin cache
docker build --no-cache -t escalafin:test .

# Build y ver logs completos
docker build --progress=plain --no-cache -t escalafin:test .

# Correr contenedor de prueba
docker run -p 3000:3000 escalafin:test
```

🔍 Comandos de Diagnóstico

Verificar archivos y permisos

```
# Ver permisos de scripts
ls -lah *.sh

# Dar permisos de ejecución
chmod +x start-improved.sh emergency-start.sh healthcheck.sh

# Ver tamaño de directorios
du -sh app/
du -sh app/node_modules/

# Ver últimas modificaciones
ls -lt | head -10
```

Verificar sincronización de dependencias

```
# Ver versión de dependencias en package.json
cat app/package.json | grep -A 5 "dependencies"

# Ver versión instalada en package-lock.json
cat app/package-lock.json | grep -A 2 "googleapis"

# Verificar integridad de package-lock.json
cd app && npm ci --dry-run
```

Generar hashes para verificación

```
# Hash del Dockerfile (para verificar versión en EasyPanel)
md5sum Dockerfile

# Hash de package.json
md5sum app/package.json

# Hash de múltiples archivos
md5sum Dockerfile app/package.json app/package-lock.json
```

🛠️ Comandos de Mantenimiento

Regenerar package-lock.json

```
cd app

# Método 1: npm install
rm package-lock.json
npm install

# Método 2: npm ci (más estricto)
rm -rf node_modules package-lock.json
npm install

# Volver al directorio principal
cd ..
```

Verificar y corregir permisos

```
# Dar permisos a TODOS los scripts
find . -name "*.sh" -exec chmod +x {} \;

# Verificar permisos
find . -name "*.sh" -exec ls -lah {} \;
```

Limpiar archivos temporales

```
# Limpiar logs antiguos
find . -name "*.log" -mtime +7 -delete

# Limpiar archivos de respaldo
find . -name "*.backup" -delete
find . -name "*~" -delete

# Ver espacio en disco
df -h
du -sh *
```

🎯 Flujos de Trabajo Completos

Flujo 1: Deploy Normal

```
# 1. Verificar que todo esté bien
./scripts/pre-deploy-verification.sh

# 2. Commit y push
git add .
git commit -m "feat: nueva funcionalidad"
git push origin main

# 3. En EasyPanel UI: clic en "Deploy"
```

Flujo 2: Deploy con Problemas de Cache

```
# 1. Diagnosticar
./scripts/cache-diagnostics.sh

# 2. Corregir problemas detectados
cd app && npm install && cd ..

# 3. Verificar nuevamente
./scripts/pre-deploy-verification.sh

# 4. Commit y push
git add .
git commit -m "fix: sincronizar dependencias"
git push origin main

# 5. En EasyPanel UI:
#    ✓ Clear build cache
#    → Rebuild
```

Flujo 3: Rebuild Completo desde Cero

```
# 1. Limpiar todo local
docker system prune -a --volumes -f

# 2. Verificar archivos
./scripts/pre-deploy-verification.sh

# 3. Build local de prueba
docker build --no-cache -t escalafin:test .

# 4. Si el build local funciona, push a GitHub
git push origin main

# 5. En EasyPanel:
#    - Detener el servicio
#    - Eliminar el contenedor
#    - Clear build cache
#    - Rebuild
```

📊 Comandos para Monitoreo

Ver logs de la aplicación

```
# Logs locales
tail -f app/server.log
tail -f dev-server.log

# Logs de Docker
docker logs <container-id>
docker logs -f <container-id>  # Seguir logs en tiempo real
```

Verificar salud de la app

```
# Probar healthcheck local
./healthcheck.sh

# Probar endpoint de salud
curl http://localhost:3000/api/health

# Ver uso de recursos
docker stats
```

Monitorear builds

```
# Ver historial de builds de Docker
docker history escalafin:latest

# Ver tamaño de capas
docker history --no-trunc --format "{{.Size}}\t{{.CreatedBy}}" escalafin:latest

# Timing de cada paso
docker build --progress=plain . 2>&1 | grep "DONE"
```

🔐 Comandos de Seguridad

Verificar secrets y variables de entorno

```
# NUNCA imprimas secrets reales, solo verifica que existan
env | grep -i "DATABASE_URL" | sed 's/=.*/=***HIDDEN***/'

# Verificar archivo .env (sin mostrar contenido sensible)
ls -lah app/.env
wc -l app/.env
```

Verificar integridad de archivos

```
# Generar checksums de archivos críticos
sha256sum Dockerfile > checksums.txt
sha256sum app/package.json >> checksums.txt
sha256sum app/package-lock.json >> checksums.txt

# Verificar checksums
sha256sum -c checksums.txt
```

🎨 Aliases Útiles (Agregar a ~/.bashrc)

```
# Agregar al final de ~/.bashrc
alias cdescala='cd /home/ubuntu/escalafin_mvp'
alias verif='./scripts/pre-deploy-verification.sh'
alias diag='./scripts/cache-diagnostics.sh'
alias glog='git log --oneline --graph --all -10'
alias gstat='git status -sb'
alias dps='docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"'
alias dlogs='docker logs -f $(docker ps -q -n 1)'

# Después de agregar, recargar:
source ~/.bashrc
```

📚 Comandos de Respaldo

Crear backup antes de cambios importantes

```
# Backup de la base de datos
./backup-db.sh

# Backup del directorio completo
tar -czf escalafin-backup-$(date +%Y%m%d-%H%M%S).tar.gz \
    --exclude=node_modules \
    --exclude=.next \
    --exclude=.git \
    /home/ubuntu/escalafin_mvp/

# Listar backups
ls -lah *.tar.gz
```

Restaurar desde backup

```
# Restaurar base de datos
./restore-db.sh

# Restaurar directorio
tar -xzf escalafin-backup-YYYYMMDD-HHMMSS.tar.gz
```

⚡ Comandos de Emergencia

Si la app no inicia

```
# 1. Ver logs
docker logs <container-id>

# 2. Entrar al contenedor
docker exec -it <container-id> /bin/sh

# 3. Verificar dentro del contenedor
ls -la /app
cat /app/package.json
node --version
npm --version

# 4. Script de emergencia
./emergency-start.sh
```

Si hay problemas con la BD

```
# Regenerar Prisma Client
cd app
npx prisma generate
npx prisma migrate dev

# Verificar conexión a BD
npx prisma db push --skip-generate
```

✅ Checklist de Comandos por Situación

Antes de CADA deploy:

```
□ ./scripts/pre-deploy-verification.sh
□ git status
□ git add .
□ git commit -m "mensaje"
□ git push origin main
```

Después de cambios en dependencias:

```
□ cd app && npm install
□ git add package-lock.json
□ ./scripts/pre-deploy-verification.sh
```

Después de cambios en Dockerfile:

```
□ docker build --no-cache .
□ ./scripts/pre-deploy-verification.sh
□ git push origin main
```

Si hay errores en EasyPanel:

```
□ ./scripts/cache-diagnostics.sh
□ Revisar y corregir problemas
□ En EasyPanel UI: Clear cache + Rebuild
```

Comandos actualizados: 29 de Octubre, 2025
Mantén este archivo a mano para referencia rápida