Salvaguarda de main en Producción antes de Deploy

Contexto

Procedimiento para crear una copia de seguridad del branch `main` actual en producción antes de realizar un deploy, permitiendo rollback inmediato si es necesario.

Variables de entorno

```
# Definir al inicio de la sesión
PROYECTO_DIR="/ruta/del/proyecto/sitelight"
FECHA_VERSION=$(date +%Y%m%d)
BACKUP_BRANCH="main_${FECHA_VERSION}"
```

Prerequisitos

- Acceso SSH al servidor de producción
- Permisos de escritura en el repositorio remoto (para push del backup)
- Git configurado correctamente

Proceso paso a paso

1. Conexión y preparación

```
# Conectar al servidor
ssh usuario@servidor-produccion

# Ir al directorio del proyecto
cd "$PROYECTO_DIR"
```

```
# Verificar que estamos en un repositorio git
git rev-parse --git-dir &>/dev/null || { echo "ERROR: No es un repositorio git"; exit 1; }
```

2. Verificación del estado actual

```
# Ver estado del working directory
git status

# Verificar rama actual
CURRENT_BRANCH=$(git branch --show-current)
echo "Rama actual: $CURRENT_BRANCH"

# Verificar si hay cambios sin commitear (CRÍTICO)
if [[ -n $(git status -s) ]]; then
    echo "ADVERTENCIA: Hay cambios sin commitear"
    git status -s
    echo "¿Continuar? (s/n)"
    read -r response
    [[ ! "$response" =~ ^[Ss]$ ]] && exit 1
fi
```

3. Actualizar main local

```
# Fetch sin merge
git fetch origin

# Cambiar a main (si no estamos ya)
git checkout main

# Verificar que estamos en sync con origin
LOCAL=$(git rev-parse main)
REMOTE=$(git rev-parse origin/main)

if [ "$LOCAL" != "$REMOTE" ]; then
    echo "main local y remoto difieren"
    echo "Local:  $LOCAL"
    echo "Remote: $REMOTE"
```

```
    echo "Actualizando..."
    git pull origin main
fi
```

4. Crear backup con fecha

```
# Definir nombre del backup con fecha de hoy
FECHA_VERSION=$(date +%Y%m%d)
BACKUP_BRANCH="main_${FECHA_VERSION}"

# Verificar que no existe ya
if git show-ref --verify --quiet "refs/heads/$BACKUP_BRANCH"; then
    echo "ADVERTENCIA: La rama $BACKUP_BRANCH ya existe"
    echo "Opciones:"
    echo "  1) Usar nombre con timestamp: main_${FECHA_VERSION}_$(date +%H%M)"
    echo "  2) Eliminar la existente y recrear"
    echo "  3) Cancelar"
    read -r option
    case $option in
        1) BACKUP_BRANCH="main_${FECHA_VERSION}_$(date +%H%M)" ;;
        2) git branch -D "$BACKUP_BRANCH" ;;
        *) exit 1 ;;
    esac
fi

# Crear la rama de backup (sin checkout)
git branch "$BACKUP_BRANCH"

# Verificar creación
git show-ref --verify "refs/heads/$BACKUP_BRANCH" || { echo "ERROR: No se pudo crear
$BACKUP_BRANCH"; exit 1; }
```

5. Pushear backup al remoto

```
# Push de la rama de backup
git push origin "$BACKUP_BRANCH"

# Verificar que está en remoto
```

```
git ls-remote --heads origin "$BACKUP_BRANCH" || { echo "ERROR: No se pudo pushear"; exit 1; }

echo "✓ Backup creado: $BACKUP_BRANCH"
```

6. Verificación del backup

```
# Ver último commit del backup
echo "Último commit en $BACKUP_BRANCH:"
git log --oneline -1 "$BACKUP_BRANCH"

# Ver todas las ramas main*
echo "Ramas de backup disponibles:"
git branch -a | grep "main_"

# Comparar backup con main actual (deben ser idénticos)
DIFF=$(git diff main "$BACKUP_BRANCH")
if [ -n "$DIFF" ]; then
    echo "ADVERTENCIA: Hay diferencias entre main y $BACKUP_BRANCH"
    echo "$DIFF"
else
    echo "✓ main y $BACKUP_BRANCH son idénticos"
fi
```

7. Deploy del nuevo main

```
# Ahora es seguro actualizar main
git pull origin main

# Verificar nuevo estado
git log --oneline -5
```

Procedimiento de Rollback

Opción A: Reset hard (reescribe historia)

```
# ADVERTENCIA: Esto reescribe la historia de main
git checkout main
git reset --hard "$BACKUP_BRANCH"

# Forzar push (requiere permisos)
git push origin main --force

# Verificar
git log --oneline -1
```

Opción B: Revert (preserva historia)

```
# Identificar commits problemáticos
git log --oneline main.."$BACKUP_BRANCH"

# Revertir commits específicos
git revert <hash-commit-problemático>

# Push normal
git push origin main
```

Opción C: Merge del backup

```
# Si hubo cambios en main que queremos deshacer
git checkout main
git merge -s ours "$BACKUP_BRANCH" -m "Rollback to $BACKUP_BRANCH"
git push origin main
```

Script automatizado

```
#!/bin/bash
# backup_main_produccion.sh

set -e  # Exit on error

PROYECTO_DIR="${PROYECTO_DIR:-/ruta/del/proyecto/sitelight}"
```

```
FECHA_VERSION=$(date +%Y%m%d)
BACKUP_BRANCH="main_${FECHA_VERSION}"

echo "=== Backup de main en Producción ==="
echo "Proyecto: $PROYECTO_DIR"
echo "Backup: $BACKUP_BRANCH"
echo

cd "$PROYECTO_DIR"

# Verificar repositorio
git rev-parse --git-dir &>/dev/null || { echo "ERROR: No es un repositorio git"; exit 1; }

# Verificar cambios sin commitear
if [[ -n $(git status -s) ]]; then
    echo "ERROR: Hay cambios sin commitear"
    git status -s
    exit 1
fi

# Actualizar main
echo "Actualizando main..."
git fetch origin
git checkout main
git pull origin main

# Verificar si backup existe
if git show-ref --verify --quiet "refs/heads/$BACKUP_BRANCH"; then
    BACKUP_BRANCH="main_${FECHA_VERSION}_$(date +%H%M%S)"
    echo "AVISO: Usando $BACKUP_BRANCH (ya existía versión diaria)"
fi

# Crear backup
echo "Creando backup: $BACKUP_BRANCH"
git branch "$BACKUP_BRANCH"

# Push backup
echo "Pusheando a remoto..."
git push origin "$BACKUP_BRANCH"
```

```
# Verificación
echo
echo "✓ Backup completado exitosamente"
echo "  Branch: $BACKUP_BRANCH"
echo "  Commit: $(git log --oneline -1 "$BACKUP_BRANCH")"
echo
echo "Ahora puedes hacer deploy con seguridad:"
echo "  git pull origin main"
echo
echo "Para rollback si es necesario:"
echo "  git reset --hard $BACKUP_BRANCH"
```

Verificaciones post-deploy

```
# Comprobar que la aplicación arrancó correctamente
systemctl status nombre-servicio

# Ver logs recientes
journalctl -u nombre-servicio -n 50 --no-pager

# Verificar conectividad/endpoints
curl -I https://dominio.com/health

# Comparar versión actual vs backup
git log --oneline "$BACKUP_BRANCH"..main
```

Limpieza de backups antiguos

```
# Listar backups de más de 30 días
git for-each-ref --format='%(refname:short) %(committerdate:iso8601)' refs/heads/main_* | \
    awk -v cutoff="$(date -d '30 days ago' +%Y-%m-%d)" '$2 < cutoff {print $1}'

# Eliminar backups locales antiguos
git branch -D main_20241001 main_20241002  # etc
```

```
# Eliminar del remoto
git push origin --delete main_20241001 main_20241002
```

Troubleshooting

"cannot push: rejected"

```
# Verificar permisos en remoto
git remote -v
git ls-remote --heads origin

# Si es necesario, forzar push del backup (solo backup, nunca main)
git push origin "$BACKUP_BRANCH" --force
```

"already exists"

```
# Ver cuándo se creó
git log --oneline -1 main_$(date +%Y%m%d)

# Opciones:
# 1. Usar timestamp completo
BACKUP_BRANCH="main_$(date +%Y%m%d_%H%M%S)"

# 2. Eliminar la existente
git branch -D main_$(date +%Y%m%d)
git push origin --delete main_$(date +%Y%m%d)
```

"divergent branches"

```
# Ver diferencias
git log --oneline --graph --all --decorate | head -20

# Forzar sincronización (solo en casos excepcionales)
git reset --hard origin/main
```

Notas importantes

- **Naming convention**: `main_YYYYMMDD` o `main_YYYYMMDD_HHMM` si hay múltiples backups en un día
- **Retención**: Mantener backups de al menos 30 días
- **Remoto**: Siempre pushear el backup al remoto (protección ante pérdida del servidor)
- **Verificación**: Confirmar que backup y main son idénticos antes del deploy
- **Rollback**: `reset --hard` es más rápido pero reescribe historia; `revert` preserva historia

---

Última actualización: 2025-11-07
Validado en: Git 2.x
Entorno: Producción sitelight

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

Revision #1
Created 2025-11-07 05:39:52 UTC by Abkrim
Updated 2025-11-07 05:42:03 UTC by Abkrim