

# Guía Profesional v5.0 – Flujo Completo con Ejemplos por Proyecto

Complemento del flujo v4.0 con ejemplos concretos para cada proyecto (Signal Mapper, Canalizaciones, control\_documental\_mvp y SSAA). Cada comando incluye una explicación breve para usar esta guía como manual universal.

## Reglas universales

- **Un repo por proyecto:** Cada carpeta/proyecto mantiene su propio historial y releases.
- **Un .venv por proyecto:** Evita conflictos de librerías. VS Code debe apuntar al intérprete de ese proyecto.
- **main = estable:** En main vive lo validado. Los tags estables se crean aquí.
- **develop = integración:** Aquí se integran branches. Cuando develop está validado, se promueve a main.

# Capítulo 1 – Flujo completo universal (para cualquier proyecto)

Receta de punta a punta: tarea → PR → main estable → tag → Release en GitHub.

Comando	Qué hace / por qué
git status	Muestra rama actual y cambios pendientes.
git checkout develop	Te cambia a develop (base de trabajo recomendada).
git pull --rebase origin develop	Trae lo último desde GitHub y mantiene historial limpio (evita merges innecesarios).
git checkout -b feature/nombre-claro	Crea rama nueva para aislar la tarea.
git diff	Revisa cambios antes de hacer commit.
git add .	Agrega cambios al stage (listos para commit).
git commit -m "feat(modulo): descripción"	Crea un snapshot trazable del cambio.
git pull --rebase origin develop	Te alinea con el remoto antes de publicar/integrar.
git push -u origin feature/nombre-claro	Sube la rama al remoto y configura seguimiento.
GitHub: Pull Request (base: develop)	Integra tu rama a develop con revisión y control.
git checkout develop	Vuelves a develop luego del merge del PR.
git pull --rebase origin develop	Actualizas develop local con lo mergeado.
git checkout main	Vas a la rama estable.
git pull --rebase origin main	Sincronizas main local con GitHub antes de promover.
git merge --no-ff develop	Promueves develop a main dejando un merge explícito (trazabilidad de release).
git push origin main	Publicas la versión estable en GitHub.
git tag -a vX.Y.Z -m "Release vX.Y.Z"	Marca el commit exacto como versión estable (tag anotado).
git push origin vX.Y.Z	Publica el tag en GitHub.
GitHub: Releases → Draft a new release	Crea el Release oficial asociado al tag, con notas y publicación.

## Capítulo 2 – Signal Mapper (conservador / estable)

Prioriza estabilidad. Taggea versiones validadas con frecuencia moderada.

**Cómo decides “estable”:** Estable = abrir app, cargar/guardar plantillas JSON, operaciones de mapeo clave sin crash.

### Comandos típicos y explicación:

Comando	Qué hace / por qué
git checkout -b feature/remap-signals	Aísla una nueva función de remapeo.
git commit -m "feat(signals): add remap rules"	Deja trazado el cambio por módulo.
git tag -a v1.2.0 -m "Signal Mapper estable v1.2.0"	Marca versión estable (solo en main).

**Esquema de versiones recomendado:** Versiones típicas: v1.2.0 (estable), v1.2.1 (hotfix), v1.3.0 (features).

# Capítulo 3 – Canalizaciones (altamente iterativo)

Iteración rápida. Usa pre-releases alpha/beta antes de llegar a v1.0.0.

**Cómo decides “estable”:** Estable = no crashea al dibujar/editar, guardar/cargar escena OK, export PDF OK.

**Comandos típicos y explicación:**

Comando	Qué hace / por qué
git checkout -b feature/canvas-background	Cambio de alto impacto (scene/view) aislado.
git commit -m "feat(canvas): add locked background image"	Snapshot para revertir rápido si algo falla.
git tag -a v0.9.0-alpha -m "Canalizaciones alpha (canvas bg)"	Pre-release para pruebas antes de estabilidad total.

**Esquema de versiones recomendado:** Versiones típicas: v0.9.0-alpha → v0.9.0-beta → v1.0.0 (estable).

# Capítulo 4 – control\_documental\_mvp (entregables / trazabilidad)

Alta trazabilidad. Tags por cada entrega/hotfix.

**Cómo decides “estable”:** Estable = carga Excel, filtros/semáforos correctos, gráficos correctos, ordenamientos OK.

## Comandos típicos y explicación:

Comando	Qué hace / por qué
git tag -a v2.6.3 -m "Entrega v2.6.3"	Marca exactamente lo entregado.
git checkout -b hotfix/fix-chart-percent	Rama de corrección urgente.
git commit -m "fix(chart): correct percent calculation"	Registra el hotfix de forma clara.
git tag -a v2.6.4 -m "Hotfix v2.6.4"	Nueva versión estable tras corrección.

**Esquema de versiones recomendado:** Versiones típicas: v2.6.3 (entrega) → v2.6.4 (hotfix) → v2.7.0 (mejora).

# Capítulo 5 – SSAA (crítico / flujo estricto)

Control total. Branches obligatorios y tags solo tras validación completa.

**Cómo decides “estable”:** Estable = persistencia OK (nombres de escenarios), cálculos consistentes, exportables OK, UI sin errores.

## Comandos típicos y explicación:

Comando	Qué hace / por qué
git checkout -b refactor/model-ui-separation	Refactor aislado (alto riesgo) sin mezclar con features.
git commit -m "refactor(model): separate UI from calculations"	Commit atómico de cambio estructural.
git tag -a v1.0.0 -m "SSAA v1.0.0 estable"	Marca versión estable validada (solo en main).

**Esquema de versiones recomendado:** Versiones típicas: v1.0.0 (estable) → v1.0.1 (hotfix) → v1.1.0 (features).

# Capítulo 6 – Plantilla universal de Release en GitHub

Para que el Release sirva siempre, usa un formato consistente de notas.

Comando	Qué hace / por qué
Título (Release title)	Usa el mismo nombre del tag, por ejemplo: v1.0.0.
Resumen	1–2 líneas: qué habilita esta versión y para quién.
Added / Changed / Fixed	Lista corta de cambios (3–10 bulletts).
Impacto / migración	Si hay cambios de estructura o config, indica pasos concretos.
Validación	Qué probaste: abrir, guardar, exportar, casos críticos.

## Ejemplo de notas:

Added: imagen base bloqueada en canvas

Fixed: crash al conectar wires desalineados

Improved: export PDF con fondo y items

Validación: crear proyecto → dibujar → guardar/cargar → export PDF