# GitFlow

Linee guida interne per la gestione dei branch Git (modello Gitflow esteso multi-release).

Questa guida descrive le prassi interne adottate dagli sviluppatori per gestire in maniera ordinata il ciclo di vita del codice sorgente tramite Git. Si basa principalmente sul modello Gitflow, ma esteso per supportare il mantenimento di più versioni in parallelo.

## Fondamenti di Gitflow

In Gitflow base esistono alcuni branch principali:

1. **master** (o main): contiene sempre il codice rilasciato in produzione.
1. **develop**: raccoglie le feature in fase di sviluppo per la prossima release.
1. **feature/**: branch temporanei per sviluppare nuove funzionalità, partono da develop e vi vengono reintegrati una volta pronti.
1. **release/**: branch dedicati a finalizzare una nuova versione, partono da develop e vengono chiusi su main (con tag) e su develop.
1. **hotfix/**: usati per correggere bug urgenti in produzione; partono da master e vengono reintegrati sia su master che su develop.

## Estensione multi-release: perché serve

Il modello Gitflow standard gestisce bene un’unica linea di sviluppo. Tuttavia, nelle aziende è spesso necessario:

1. continuare a rilasciare correzioni o nuove feature per versioni precedenti (es. 4.7.x) mentre si lavora alla nuova (es. 4.8.0).
1. garantire stabilità e manutenzione a lungo termine per clienti che non aggiornano subito alla major release più recente.

Per questo introduciamo i branch di **maintenance/**.

## Gestione dei branch in multi-release

### Branch maintenance/

Ogni versione maggiore/minore mantenuta ha un proprio branch maintenance/[major].[minor].x.

Esempio: maintenance/4.7.x, maintenance/4.6.x, ecc.

Da maintenance/4.7.x vengono creati i branch release/4.7.9, release/4.7.10, ecc.

Una volta rilasciata la versione, il branch release/ viene chiuso e reintegrato su maintenance/4.7.x.

### Branch release/

Usati per preparare una specifica release (es. release/4.7.9).

Quando la release è pronta:

Si crea un tag (es. v4.7.9).

Si reintegra su maintenance/4.7.x.

Si chiude il branch release/4.7.9.

Si apre il corrispondente hotfix/4.7.9.x per future patch.

⚠ Nota bene: se ci sono più branch release/ attivi (es. release/4.7.10 e release/4.7.11), è obbligatorio chiudere/taggare in ordine crescente. Per esempio, per chiudere release/4.7.11 bisogna prima chiudere e reintegrare release/4.7.10.

### Branch hotfix/

Prima usavamo hotfix/4.7.9 per contenere i fix della 4.7.8 (che sarebbero comunque finiti nella successiva 4.7.9)

Ora, per maggiore chiarezza, utilizziamo il formato hotfix/4.7.8.x:

Ogni incremento del quarto numero corrisponde a una patch minore: si parte da 1 e si sale.

I fix sviluppati qui vanno riportati anche nel branch release/4.7.9 (se ancora aperto).

Quando un release/ viene chiuso, si apre sempre il corrispondente hotfix/ di mantenimento (es. hotfix/4.7.9.x).

### Esempio pratico

Stiamo sviluppando la nuova versione 4.8.0 su develop.

Alcuni clienti utilizzano ancora la 4.7.x, quindi:

Da maintenance/4.7.x si apre release/4.7.9.

Una volta pronta, si tagga v4.7.9, si reintegra su maintenance/4.7.x e si chiude.

Viene creato hotfix/4.7.9.x per eventuali bugfix.

Se serve un nuovo rilascio, apriamo release/4.7.10 da maintenance/4.7.x e ripetiamo il ciclo.

**Raccomandazioni operative**

Naming chiaro e coerente: sempre usare i prefissi (feature/, release/, maintenance/, hotfix/) per distinguere i branch.

Tag obbligatori: ogni rilascio ufficiale va sempre taggato.

Backporting: i fix su versioni precedenti vanno riportati su develop e sulle versioni successive, per evitare divergenze di codice.

Ordine di chiusura: rispettare la sequenza dei release/ per evitare conflitti e merge complessi.

Documentazione: ogni chiusura di release e apertura di hotfix va tracciata nel changelog interno.

# Versioni nei pom.xml

Con un branching model **multi-release** è fondamentale tenere versioni “parlanti” e prevedibili nei vari `pom.xml` dei vari branch.

Questo serve a non avere conflitti di artifact nello stesso repo Maven/Artifactory.

## Regole generali

1. **Usa SemVer (+ eventuale 4° numero per patch urgenti)**
   - MAJOR.MINOR.PATCH[-QUALIFIER]
   - Se avete bisogno del 4° numero per hotfix rapidi: MAJOR.MINOR.PATCH.PATCH2 (Maven lo supporta)
2. **-SNAPSHOT**, **-HOTFIX**, **-RELEASE**, **-MAINTENANCE** in base al branch
   - Nessun suffisso `-SNAPSHOT` nei commit taggati
3. **Un’unica sorgente di verità**
   - Usa un parent POM con `<version>` centrale
   - Nei moduli figli utilizza `${project.version}` o un BOM in `<dependencyManagement>`
4. **Bump automatico dopo il tag**
   - Dopo aver taggato una release, il branch che continua deve passare subito alla prossima `-SNAPSHOT`

## Versioni nei branch

| Branch | Versione in pom.xml | Note operative |
|---|---|---|
| **master/main** | versioni rilasciate (es. 4.8.0, 4.7.8) | Solo commit taggati, no suffissi - |
| **develop** | 4.8.0-SNAPSHOT → poi 4.9.0-SNAPSHOT | Dopo apertura release, bump immediato |
| **maintenance/4.7.x** | patch corrente in MAINTENANCE (es. 4.7.8-MAINTENANCE ) | Mantiene la linea viva |
| **release/4.7.9** | 4.7.9-RELEASE → al tag 4.7.9 | Dopo il tag si chiude su maintenance, che diviene 4.7.9-MAINTENANCE |
| **hotfix/4.7.9.x** | 4.7.9.1-HOTFIX → tag 4.7.9.1 → poi 4.7.9.2-HOTFIX | Ogni patch incrementa il 4° numero |

## Esempio Parent POM

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>it.example</groupId>
  <artifactId>prodotto-parent</artifactId>
  <version>4.7.10-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>core</module>
    <module>webapp</module>
  </modules>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>it.example</groupId>
        <artifactId>prodotto-bom</artifactId>
        <version>${project.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
</project>
```

## Esempio Modulo figlio

```
<project>
  <parent>
    <groupId>it.example</groupId>
    <artifactId>prodotto-parent</artifactId>
    <version>4.7.10-SNAPSHOT</version>
  </parent>

  <artifactId>webapp</artifactId>

  <dependencies>
    <dependency>
      <groupId>it.example</groupId>
      <artifactId>core</artifactId>
      <version>${project.version}</version>
    </dependency>
  </dependencies>
</project>
```

### Flusso tipico (4.7.x parallelamente a 4.8.0)

1. develop: 4.8.0-SNAPSHOT
2. Apri release/4.8.0 → su release: 4.8.0-RELEASE, su develop bump a 4.9.0-SNAPSHOT
3. maintenance/4.7.x: 4.7.8-MAINTENANCE
4. Apri release/4.7.9 → 4.7.9-RELEASE
5. Tag v4.7.9 → chiudi il branch → merge su maintenance/4.7.x
6. maintenance/4.7.x va a 4.7.9-MAINTENANCE
7. Hotfix urgente: apri hotfix/4.7.9.x con 4.7.9.1-HOTFIX → tag v4.7.9.1 → bump a 4.7.9.2-HOTFIX

Se ci sono più release contemporanee (es. release/4.7.10 e release/4.7.11) chiudere e taggare prima la più bassa.

### Comandi utili

Impostare versione su tutti i moduli:

```
mvn -q versions:set -DnewVersion=4.7.10-RELEASE -DprocessAllModules=true -DgenerateBackupPoms=false
```

Taggare:

```
mvn -q versions:set -DnewVersion=4.7.9 -DprocessAllModules=true -DgenerateBackupPoms=false
git commit -am "pom.xml version 4.7.9"
git tag v4.7.9
```

Bump dopo il tag:

```
mvn -q versions:set -DnewVersion=4.7.10-RELEASE -DprocessAllModules=true -DgenerateBackupPoms=false
git commit -am "pom.xml bump to 4.7.10-RELEASE"
```

## Versioni nel gwRegistry

Nel gwRegistry ci sono due liste con versioni da tenere allineate:

1. **gwVersionList**: versione che deve contenere oltre al fisso 4., solo .[MAJOR].[MINOR] seguito poi da un -[DESCRITTIVO_FUNZIONALE] (-SNAPSHOT, -RELEASE) Viene utilizzato per gestire la migrazione dei metadati ⇒ qui NON va esplicitato il quato numero .[FIX]
2. **gwVersionToShowList**: versione è destinata alla sola visualizzazione. oltre .[MAJOR].[MINOR] può esserci un .[FIX] nei branch di -hotfix/

L'idea è di seguire le stesse convenzioni della versione del pom.xml, con un unica eccezione per il gwVersionList.