

Code Management and Deployment

Gitflow

A Git branching model

DotBrains

GuardRails

20240802

Requirements

Code Management and Deployment

DotBrains | GuardRails

1 Explicitly Versioned

Use Semantic Versioning to identify software release version and type

2 Multiple Versions

Support multiple deployed versions

3 Release Schedules

Major releases have deadlines

4 Feature Release

Release features to multiple environments

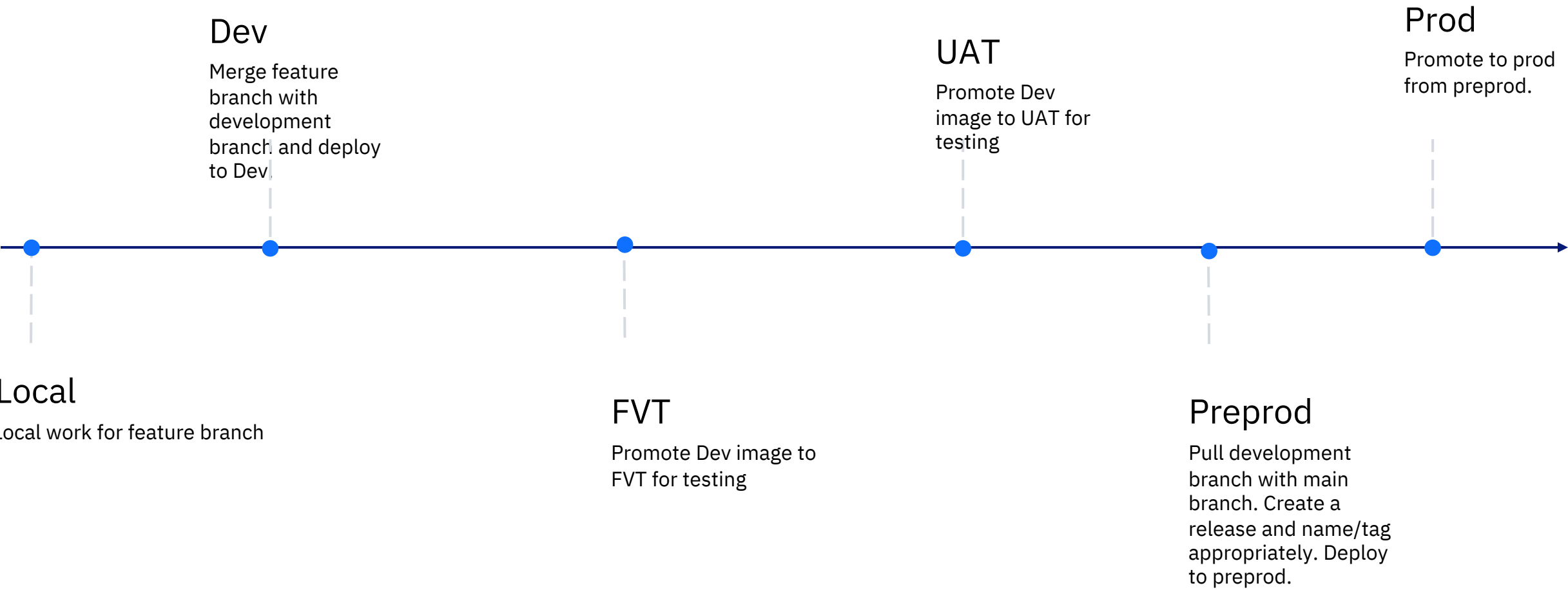
5 Hotfixes

Quickly patch production releases

6 Controls

Have control checks to manually deploy versions to specific environments

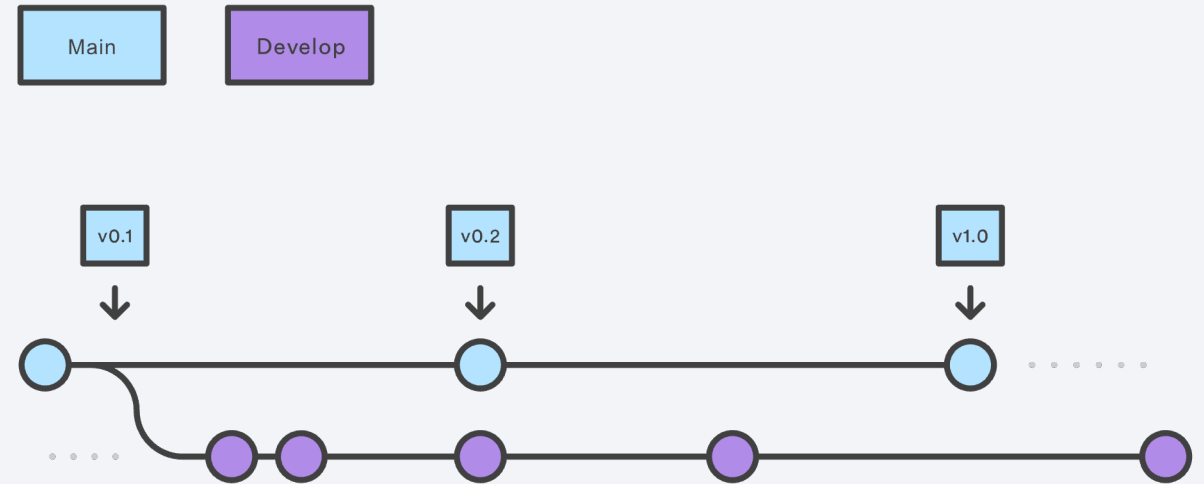
Environments



Main Branches

Main

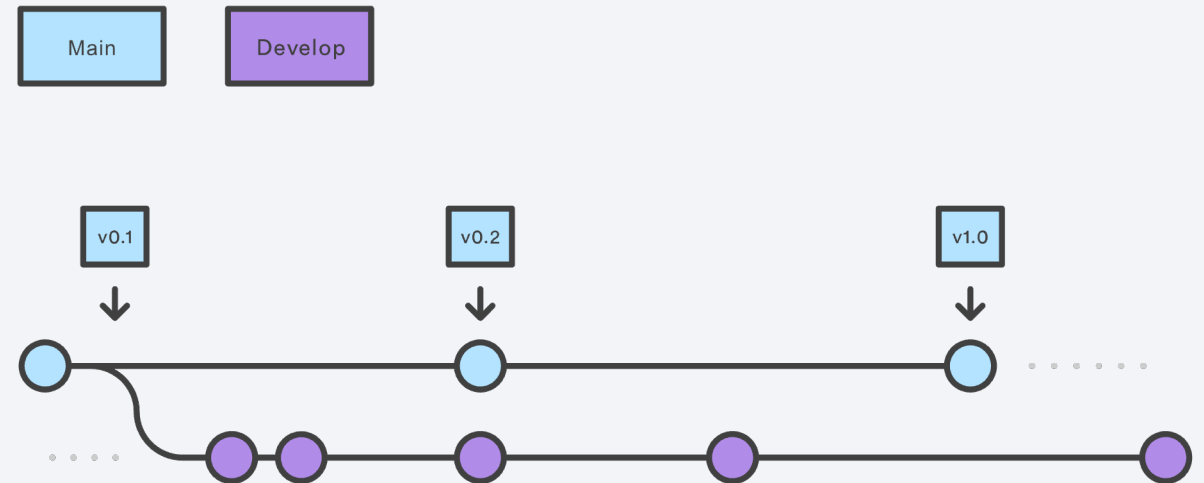
- Always represents a production-ready state
- Production-ready releases are made from this branch
- Nothing is merged into Main until we are ready to prepare for a production deployment



Main Branches

Develop

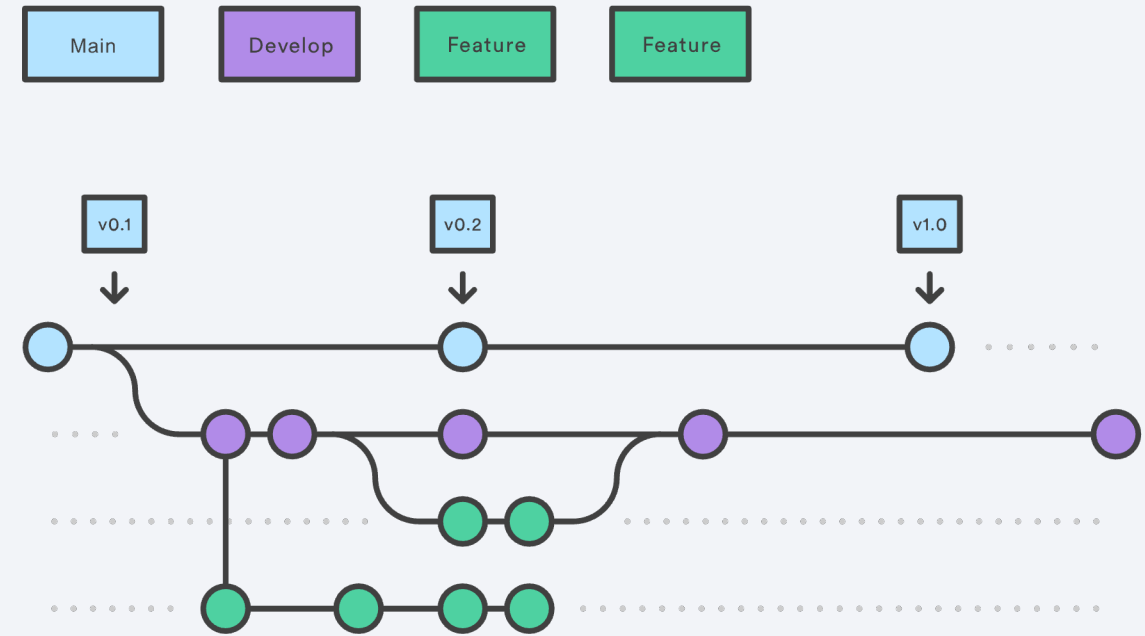
- Represents latest development changes for next release
- Automatic builds come from here
- When develop is stable and ready to be released, merge into Main and then tag.



Supporting Branches

Feature

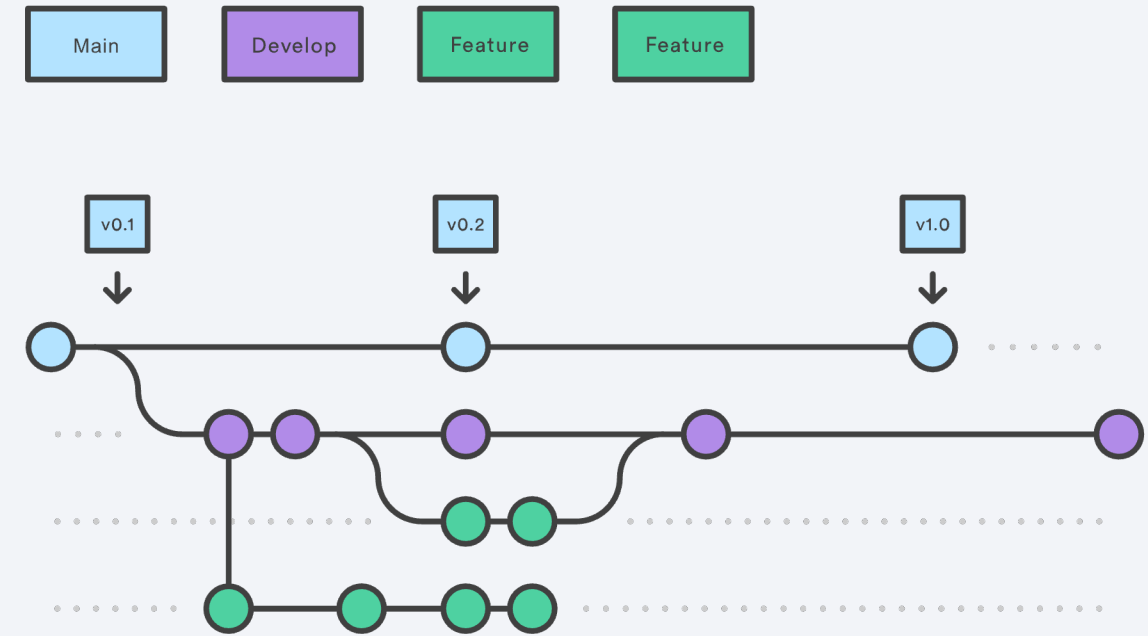
- Used to develop new features for upcoming release



Supporting Branches

Feature

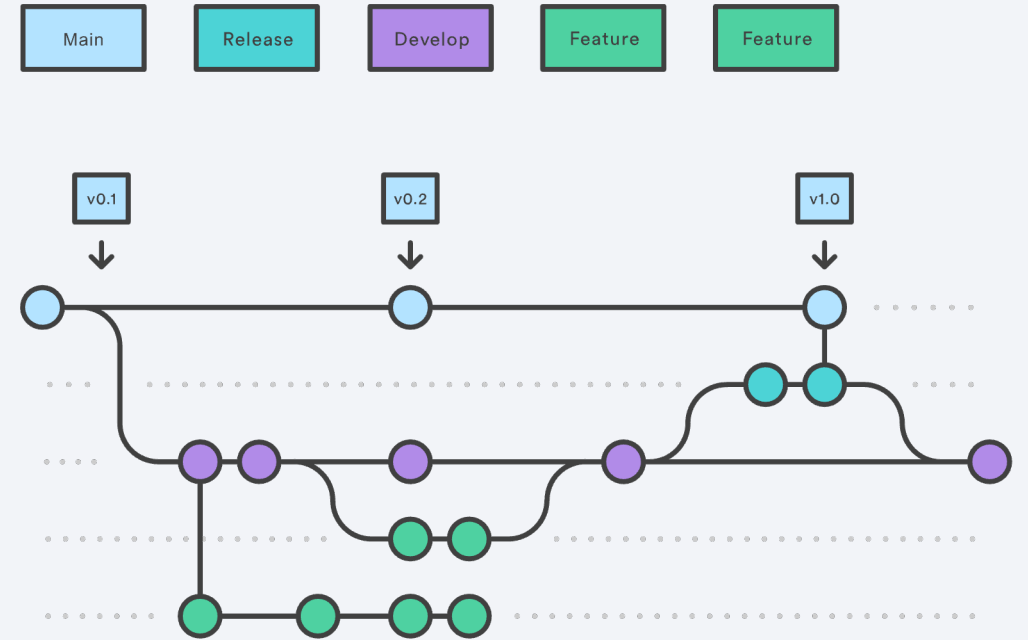
- Used to develop new features for upcoming release
- Branched from develop
- Merge only into develop, never main
- Naming convention: anything but main, develop, release-*, or hotfix-* i.e. feature-*, FEATURE/*



Supporting Branches

Release

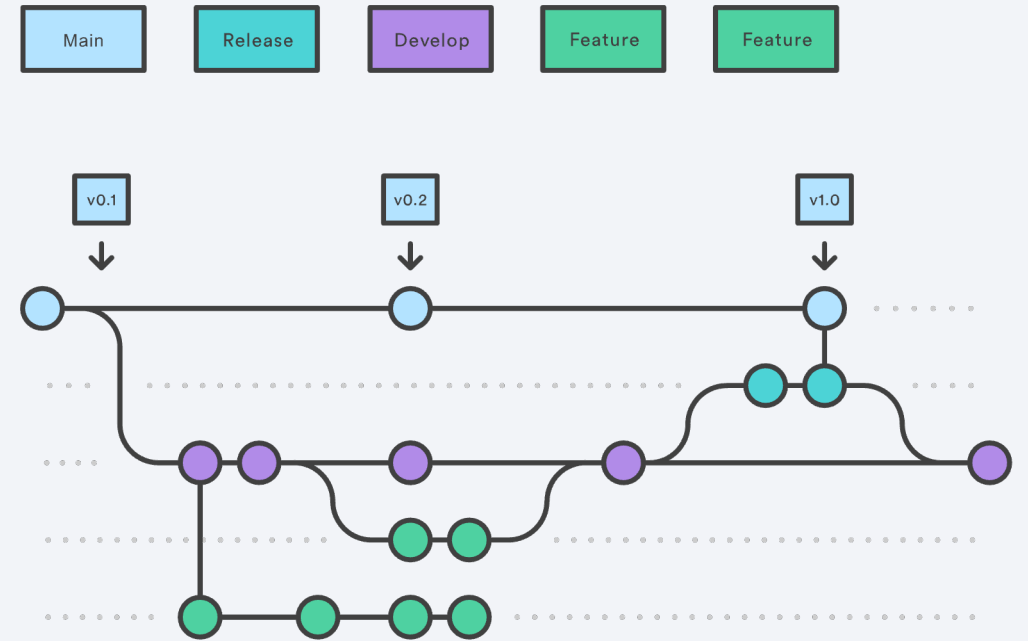
- Support preparation of new production release
- Create this branch when all targeted features present in develop. No more, no less.
- Last minute fixes, preparing metadata
- Develop branch can continue future release work



Supporting Branches

Release

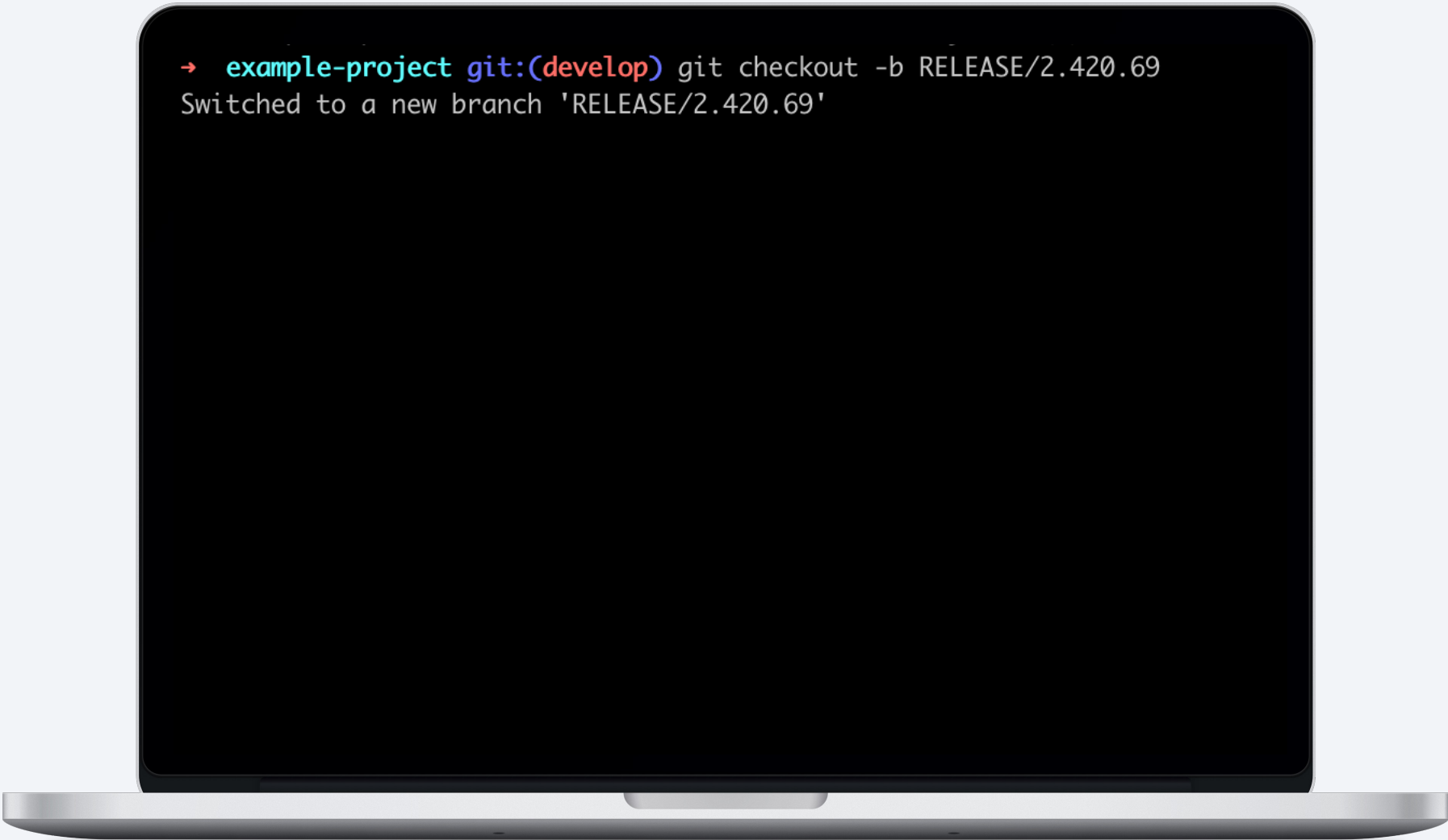
- Support preparation of new production release
- Create this branch when all targeted features present in develop. No more, no less.
- Last minute fixes, preparing metadata
- Develop branch can continue future release work
- Branched from develop
- Merge into both develop and main
- Naming convention: release-*, RELEASE/*



Git Release Commands

Code Management and Deployment

DotBrains | GuardRails

A laptop is shown from a front-facing perspective, with its screen displaying a terminal window. The terminal has a dark background with light-colored text. The text shows a command to checkout a new branch and the resulting output.

```
→ example-project git:(develop) git checkout -b RELEASE/2.420.69  
Switched to a new branch 'RELEASE/2.420.69'
```

Git Release Commands

Code Management and Deployment

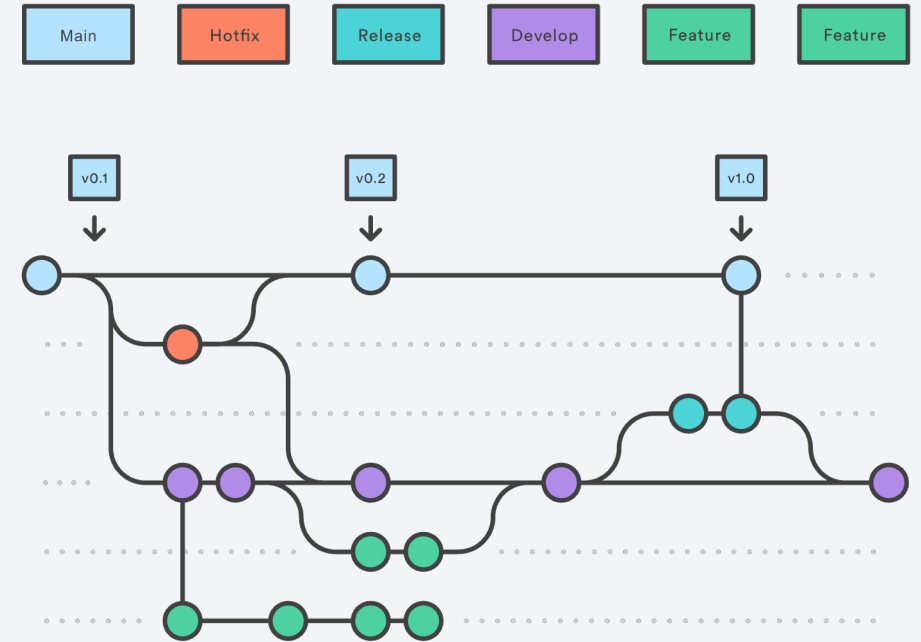
DotBrains | GuardRails

```
→ example-project git:(RELEASE/2.420.69) git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
→ example-project git:(main) git merge RELEASE/2.420.69
Updating c7711ef..d8b2747
Fast-forward
 README.md | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
→ example-project git:(main) █
```

Supporting Branches

Hotfix

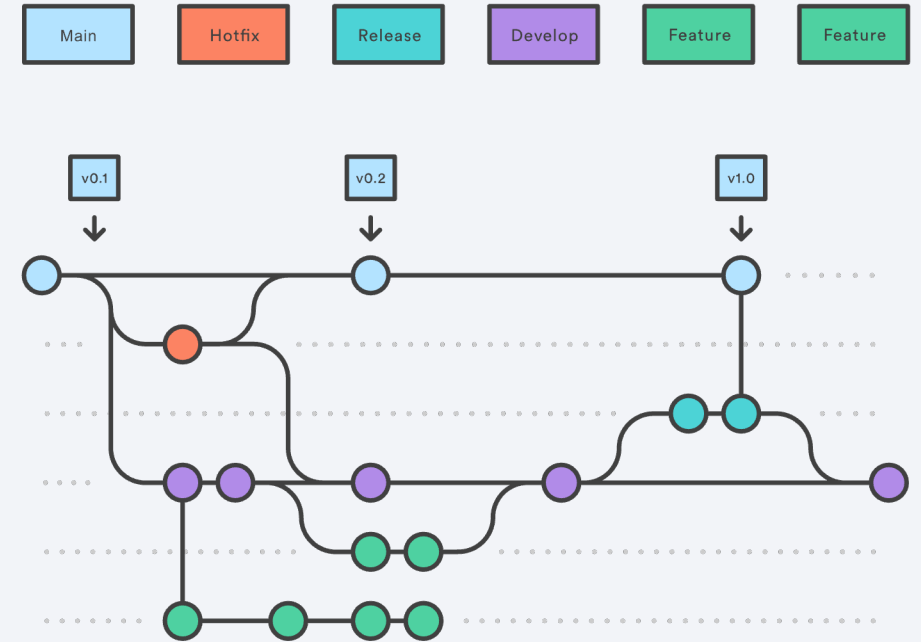
- Quickly patch production releases



Supporting Branches

Hotfix

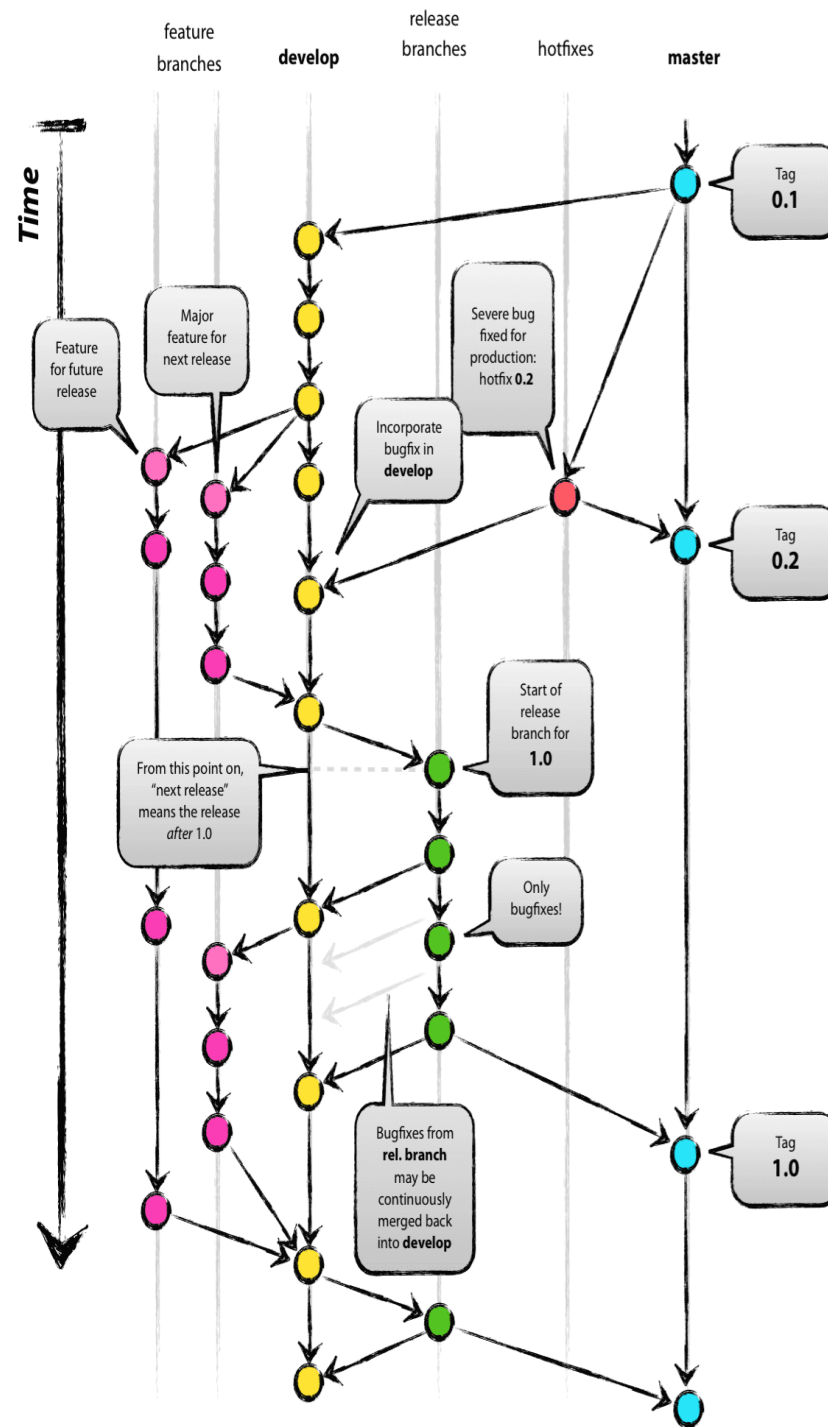
- Quickly patch production releases
- Branched from main tag that's in production
- Merge into both develop and main
- Tag main with an updated version number
- Naming convention: hotfix-*, HOTFIX/*



Gitflow

Great for release-based software

Offers a dedicated channel for hotfixes to production



Overall Flow

- 1 A develop branch created from main
- 2 A release branch created from develop
- 3 Feature branches are created from develop
- 4 When a feature is complete it is merged into the develop branch
- 5 When a release branch is done it is merged into develop and main
- 6 If an issue in main is detected a hotfix branch is created from main
- 7 Once the hotfix is complete it is merged to both develop and main

Deployment - Develop

Travis

- Pipeline triggered by new commits to develop branch
- Build, tag, and push to Cirrus
 - `example-project:develop`
 - `example-project:develop-2.2.1-${commit-hash}`

Cirrus

- Develop environment auto-deploys when new image pushed

Deployment – Testing (FVT)

Travis

- Pipeline triggered by new pre-production release tag
- Build, tag, and push to Cirrus
 - `example-project:2.2.1-alpha.001`

Cirrus

- Testing environment environment auto-deploys when new image pushed

Deployment – Staging (UAT)

Travis

- No action

Cirrus

- Staging environment manually changes application image when testing is passed

Deployment – Preprod

Travis

- Pipeline triggered by new release
- Build, tag, and push to Container Registry
 - DotBrains/example-project:2.2.1

Cirrus

- Preprod environment auto-deploys when new image pushed

Deployment – Prod

Travis

- No action

Cirrus

- Prod deployment when image version is manually changed

- <https://nvie.com/posts/a-successful-git-branching-model>
- <https://atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- <https://semver.org>

Thank you

