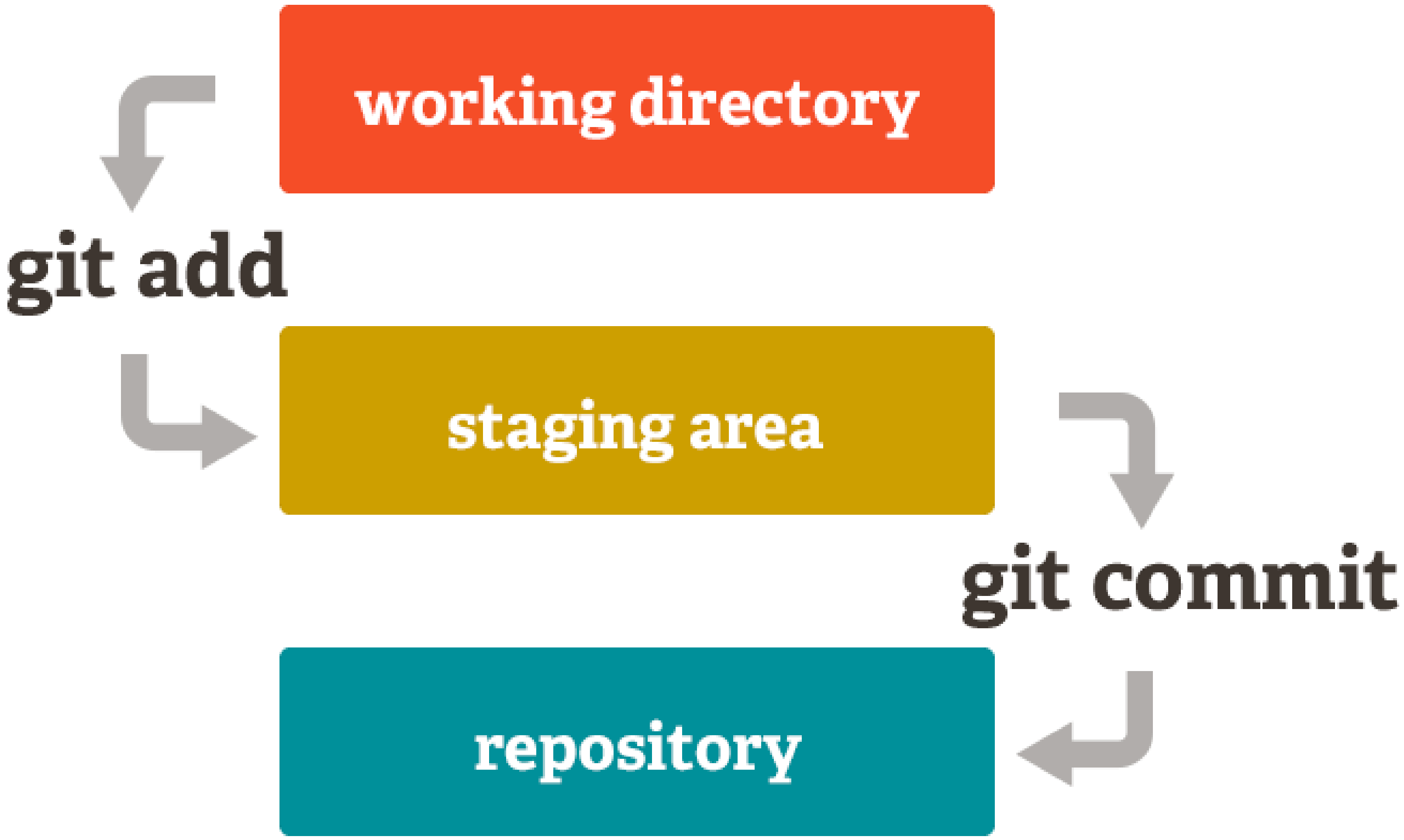
Git workflows

Agenda:

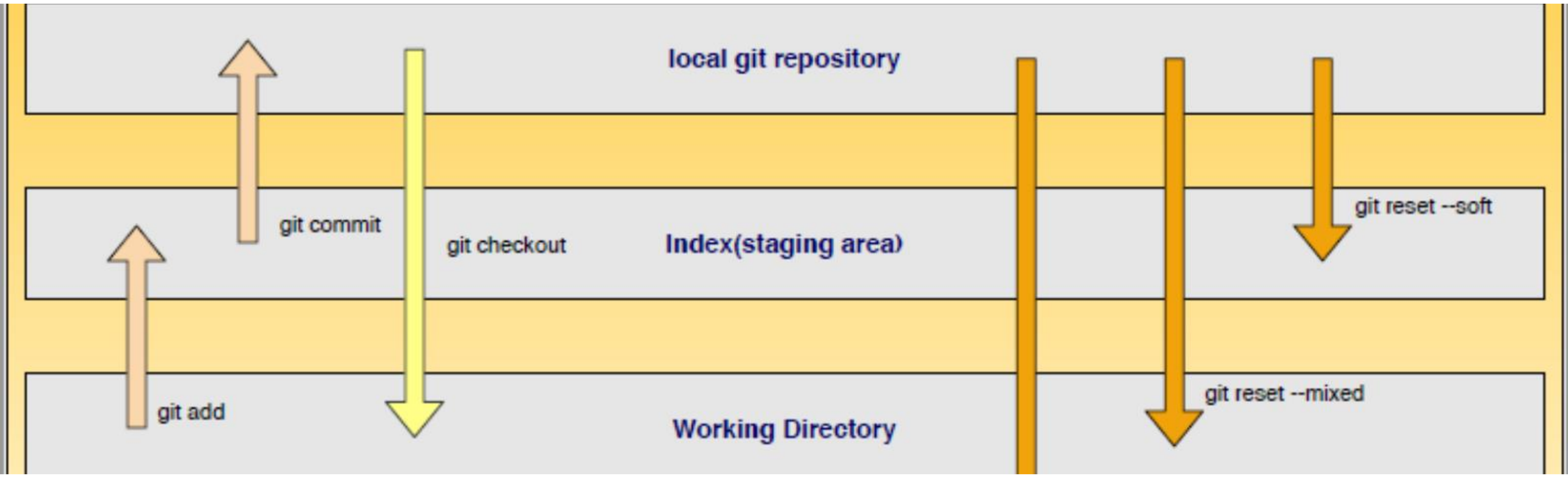
- Git workflow: Working with **Local Repository**
- Git workflow: Working with **Central Repository**
- Git workflow: Working with **Feature Branch**
- Git workflow: Working with **Gitflow**
- Git workflow: Working with **Forking**

Git workflow: Working with **Local Repository**

Git workflow: Working with **Local Repository**

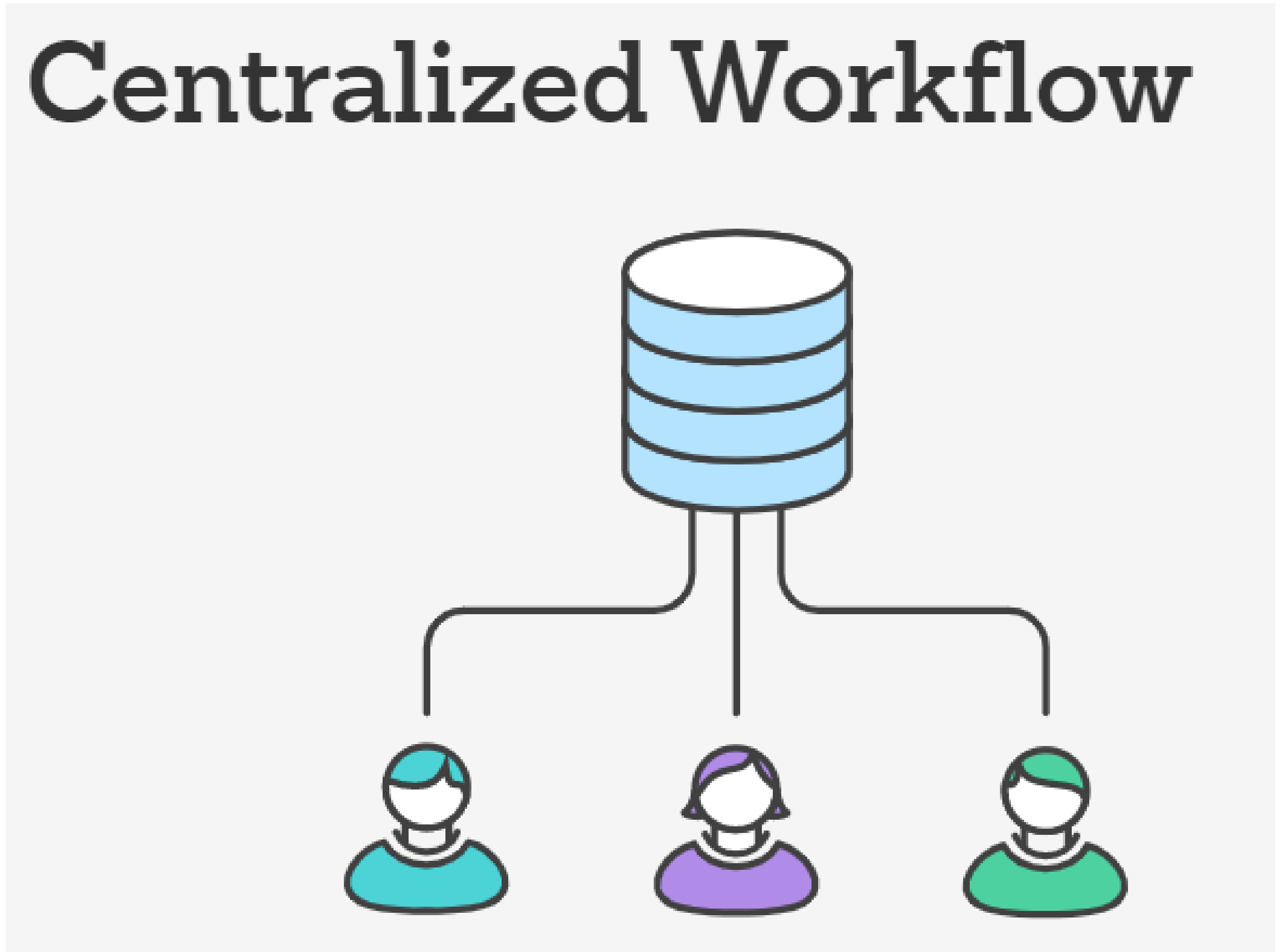


Git workflow: Working with **Local Repository**

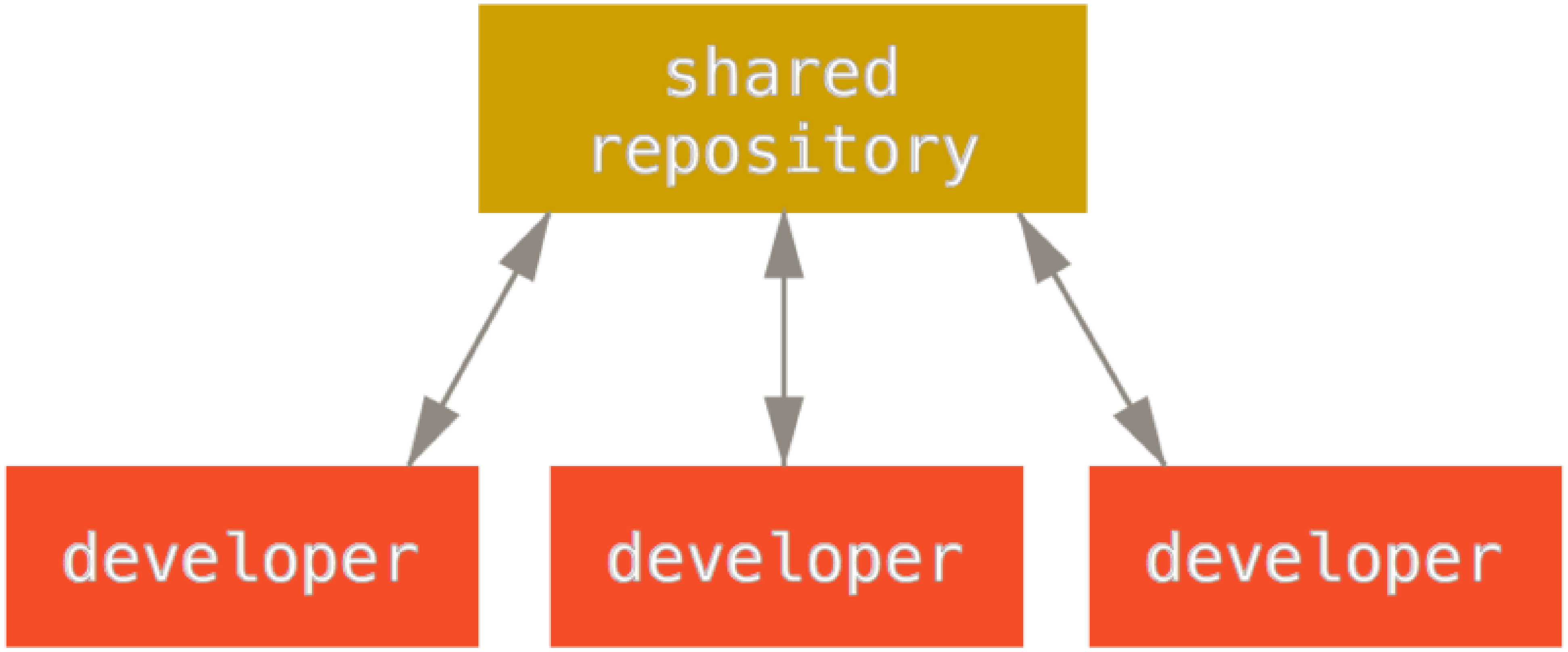


Git workflow: Working with Central Repository

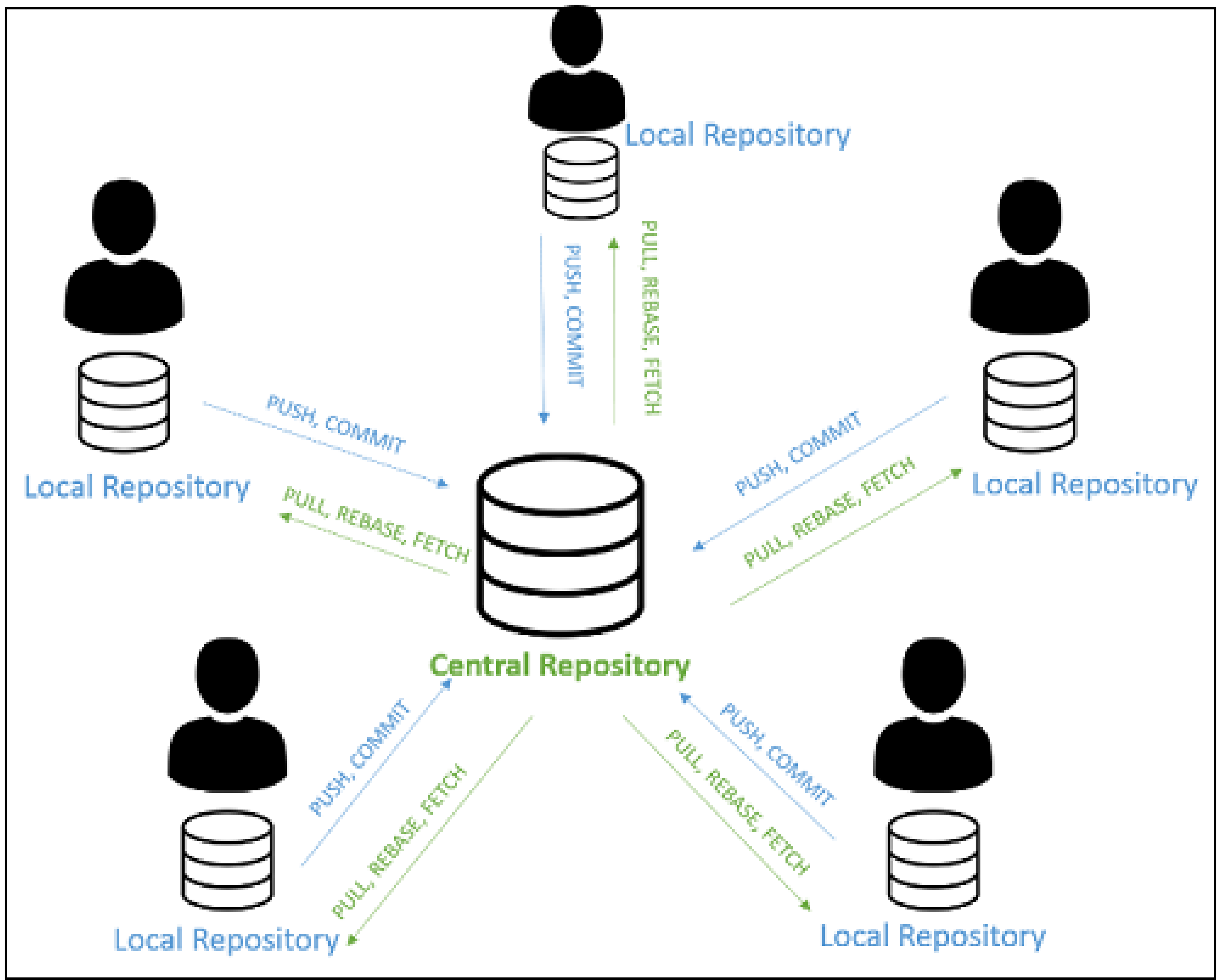
Git workflow: Working with **Central Repository**



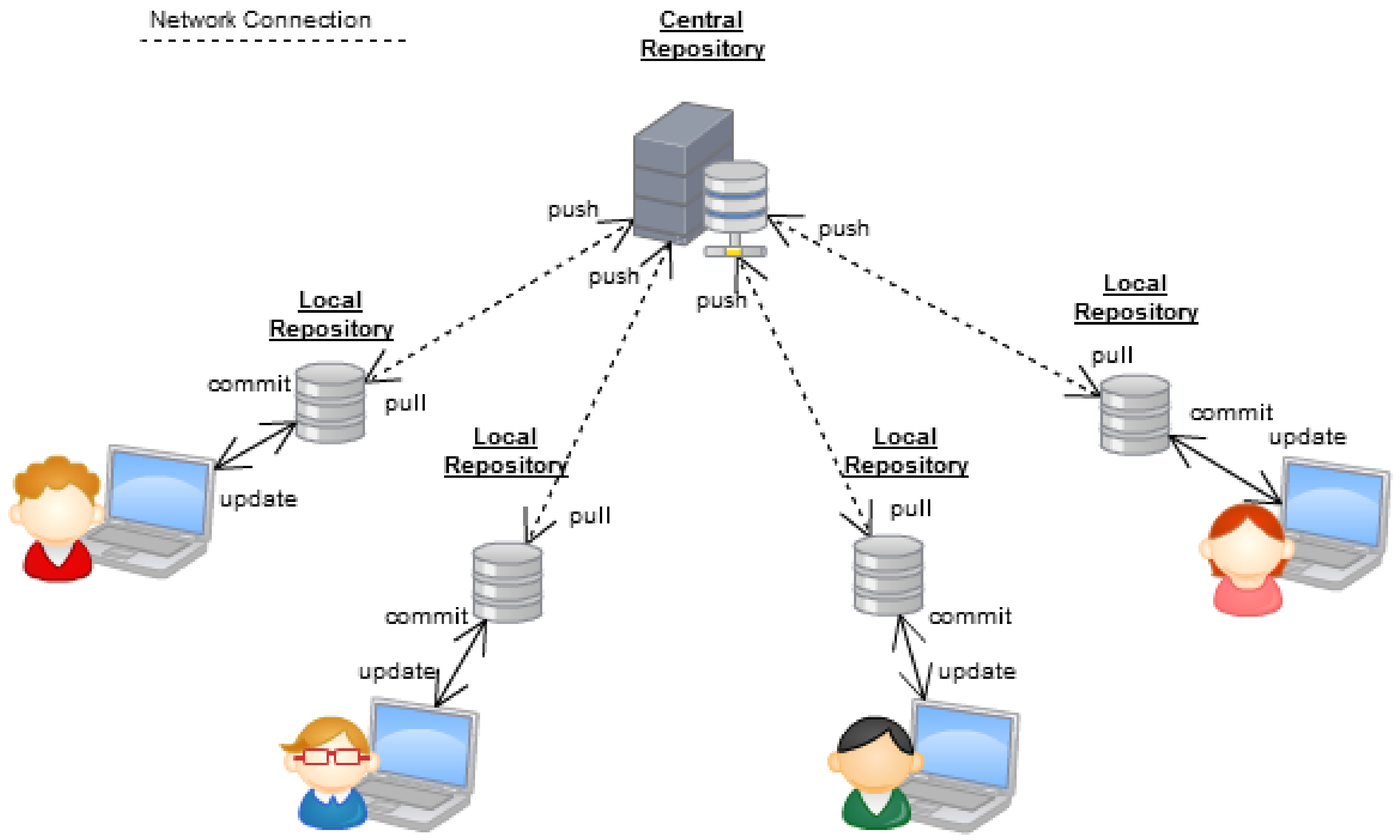
Git workflow: Working with **Central Repository**



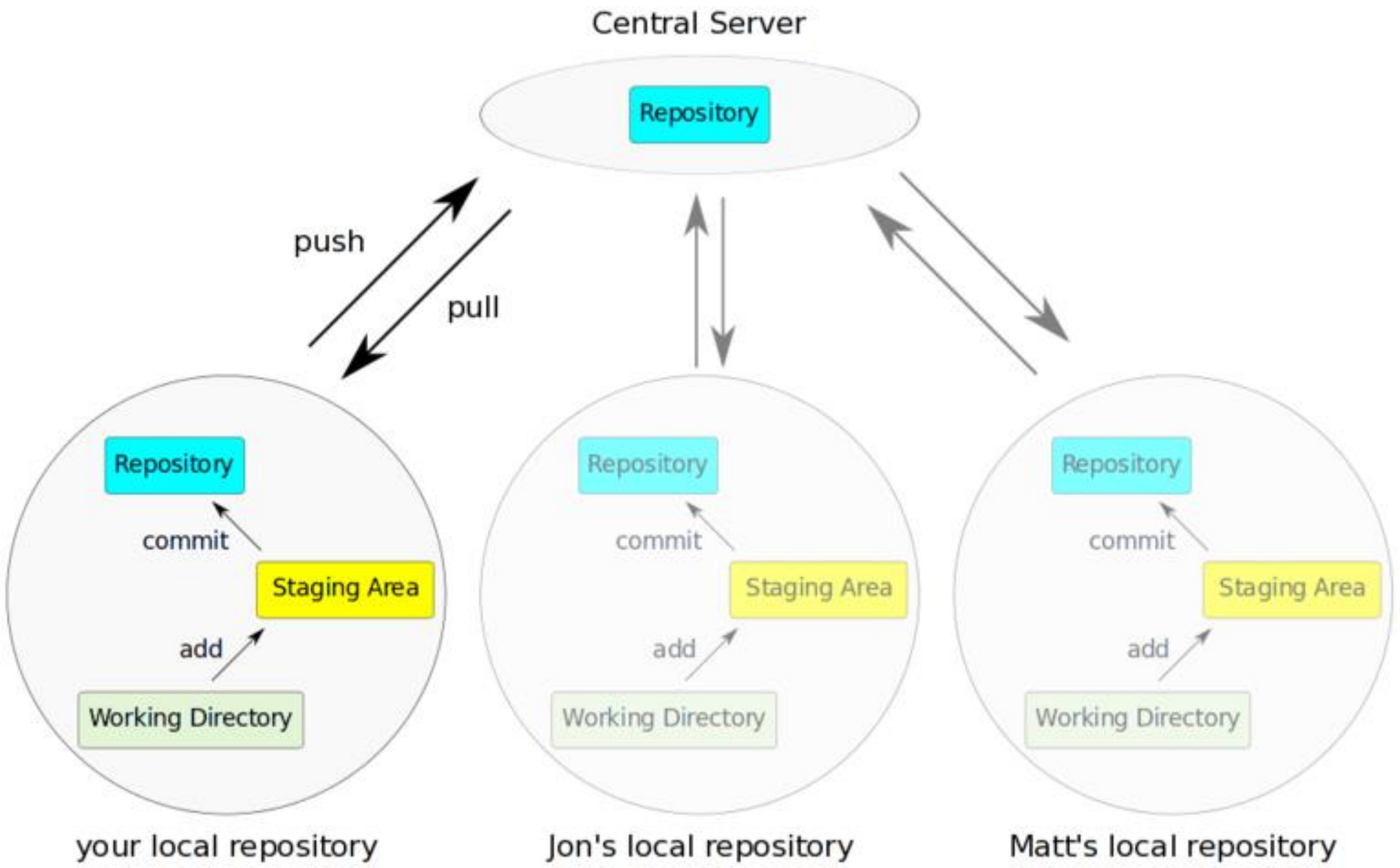
Git workflow: Working with **Central Repository**



Git workflow: Working with **Central Repository**



Git workflow: Working with **Central Repository**



Git workflow: Working with **Feature Branch**

AKA **Git pull request workflow**

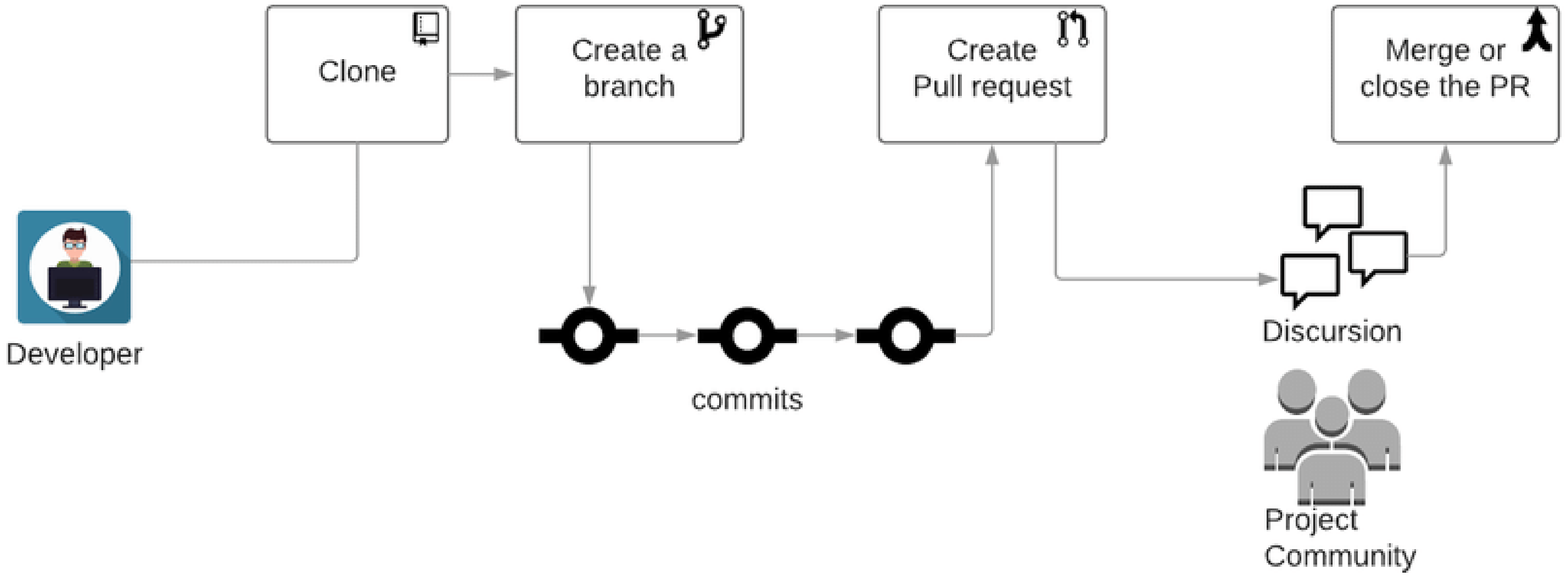
Git workflow: Working with **Feature Branch**

The Feature Branch Workflow still uses a central repository, and master still represents the official project history. But, instead of committing directly on their local master branch, developers create a new branch every time they start work on a new feature. Feature branches should have descriptive names as login-template-header, login-http-resource, refactoring-login-service, etc.

Git workflow: Working with Feature Branch

Now youre team is working alongside of the remote master branch, everyone is working on a feature branch. When each feature is done and are up to put on development the branch should be merged with remote master by a **Pull Request.**

Working with **Feature Branch: Pull Request**



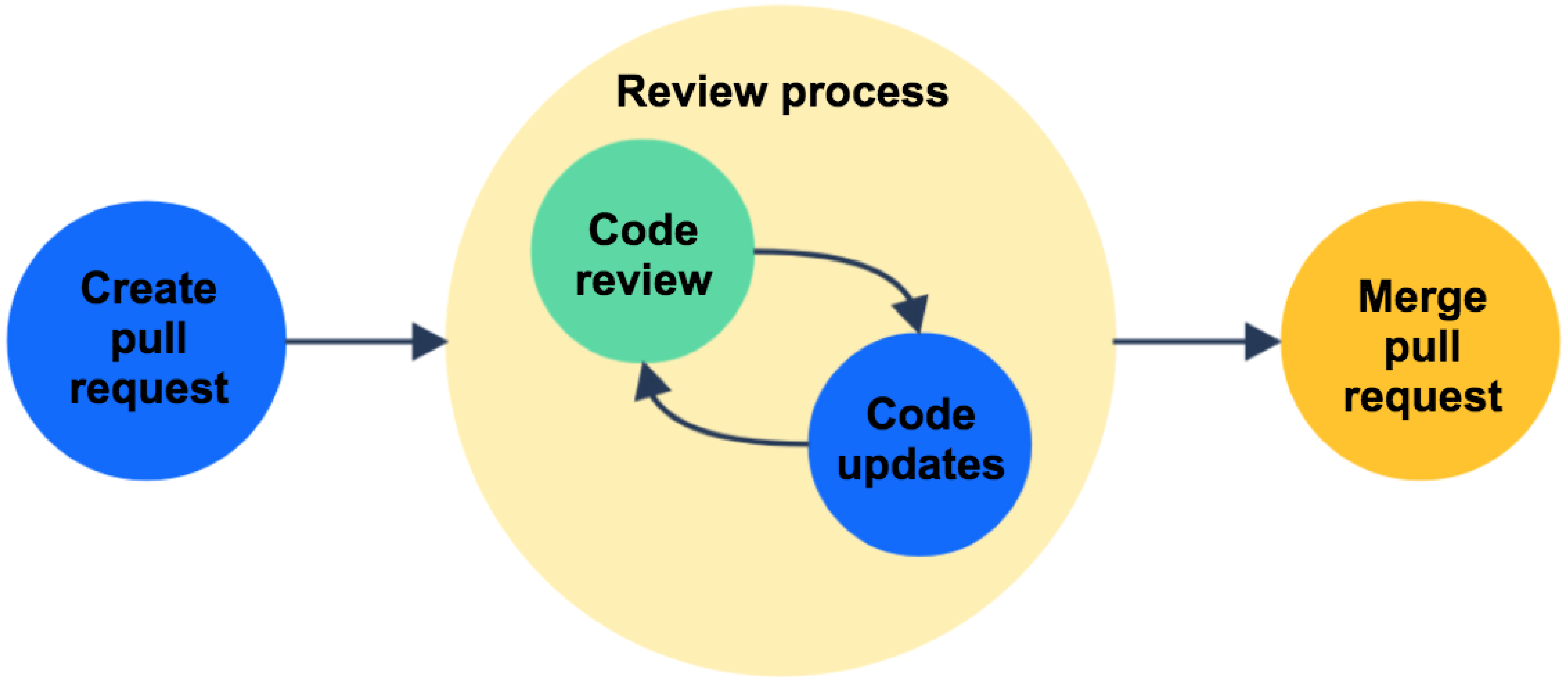
Working with **Feature Branch: Pull Request**

Pull requests are a feature that makes it easier for developers to collaborate using a git client (GitHub, Bitbucket, Stash, etc.). They provide a user-friendly web interface for discussing proposed changes before integrating them into the official project.

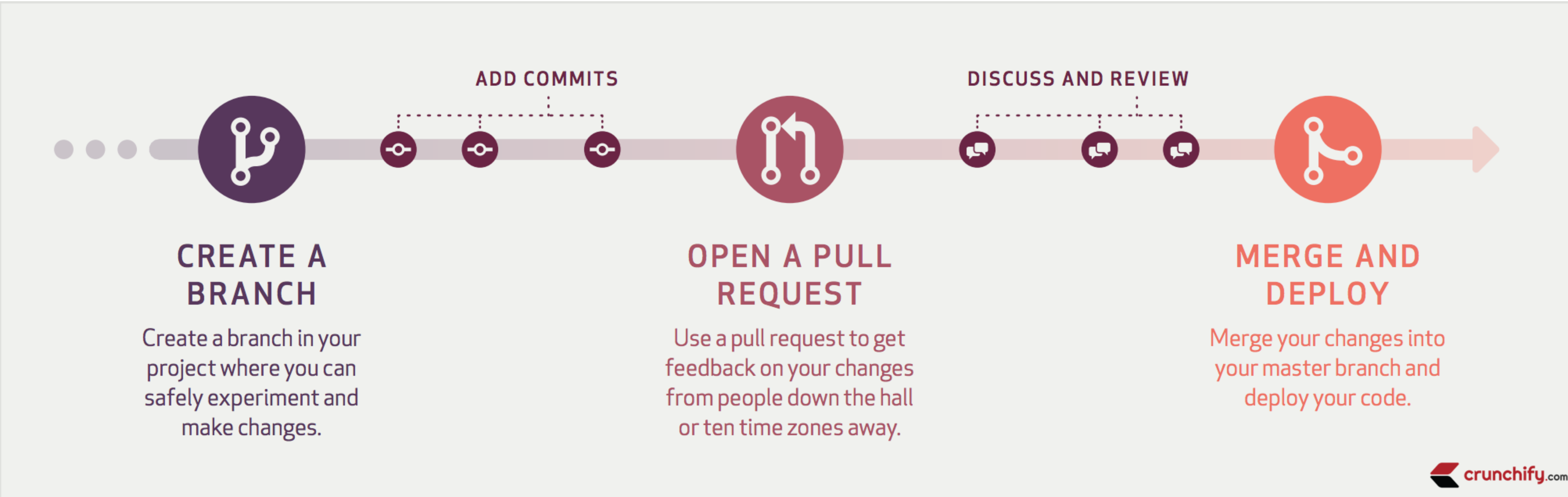
Pull request is a dedicated forum for discussing the proposed feature. If there are any problems with the changes, teammates can post feedback in the pull request and even tweak the feature by pushing follow-up commits. All of this activity is tracked directly inside of the pull request.

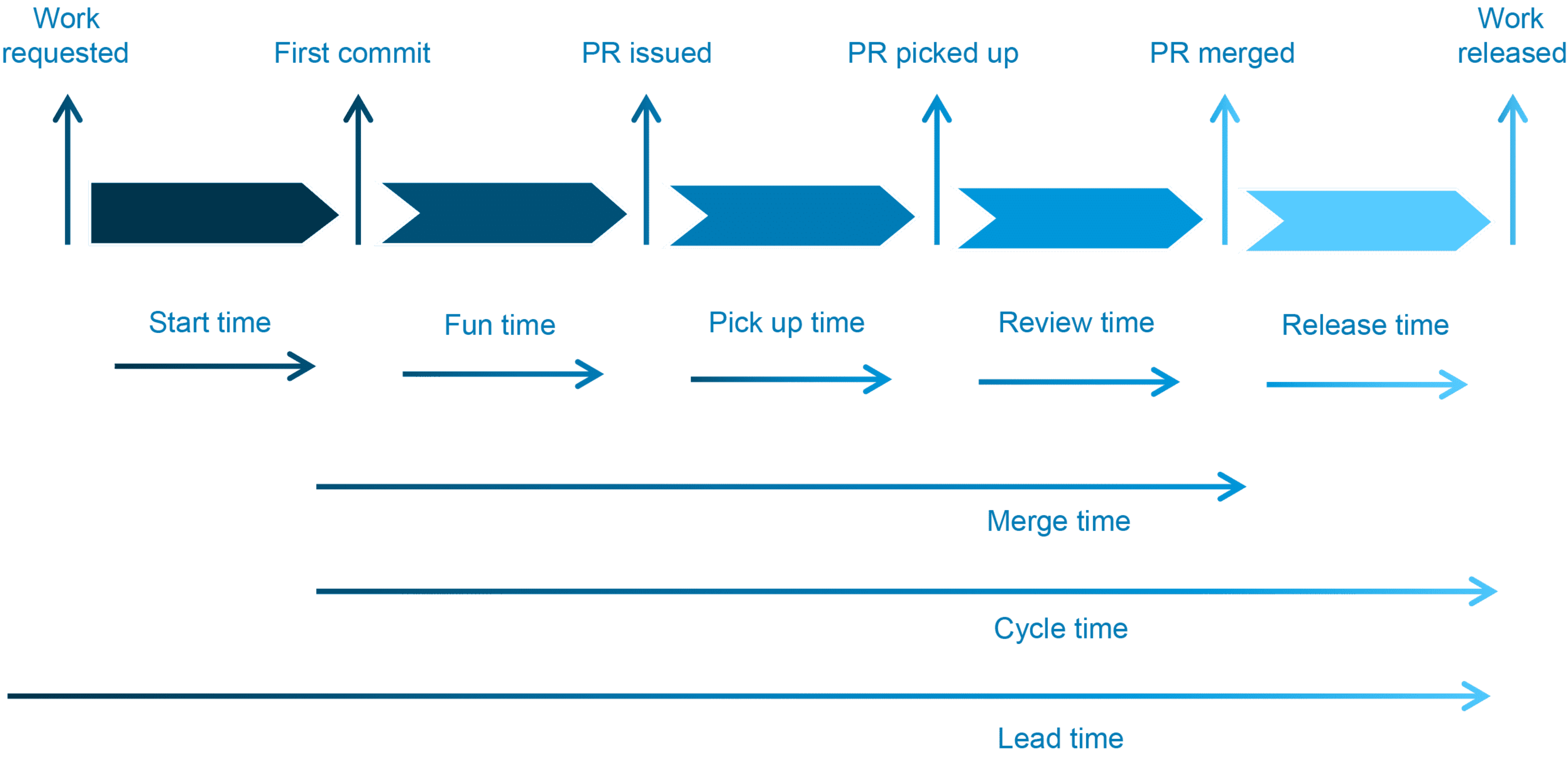
Essentially a PR should be closed after a Code Review with others developers. Code review is a major benefit of pull requests. You can think of pull requests as a discussion dedicated to a particular branch.

Working with Feature Branch: Pull Request



Working with Feature Branch: Pull Request

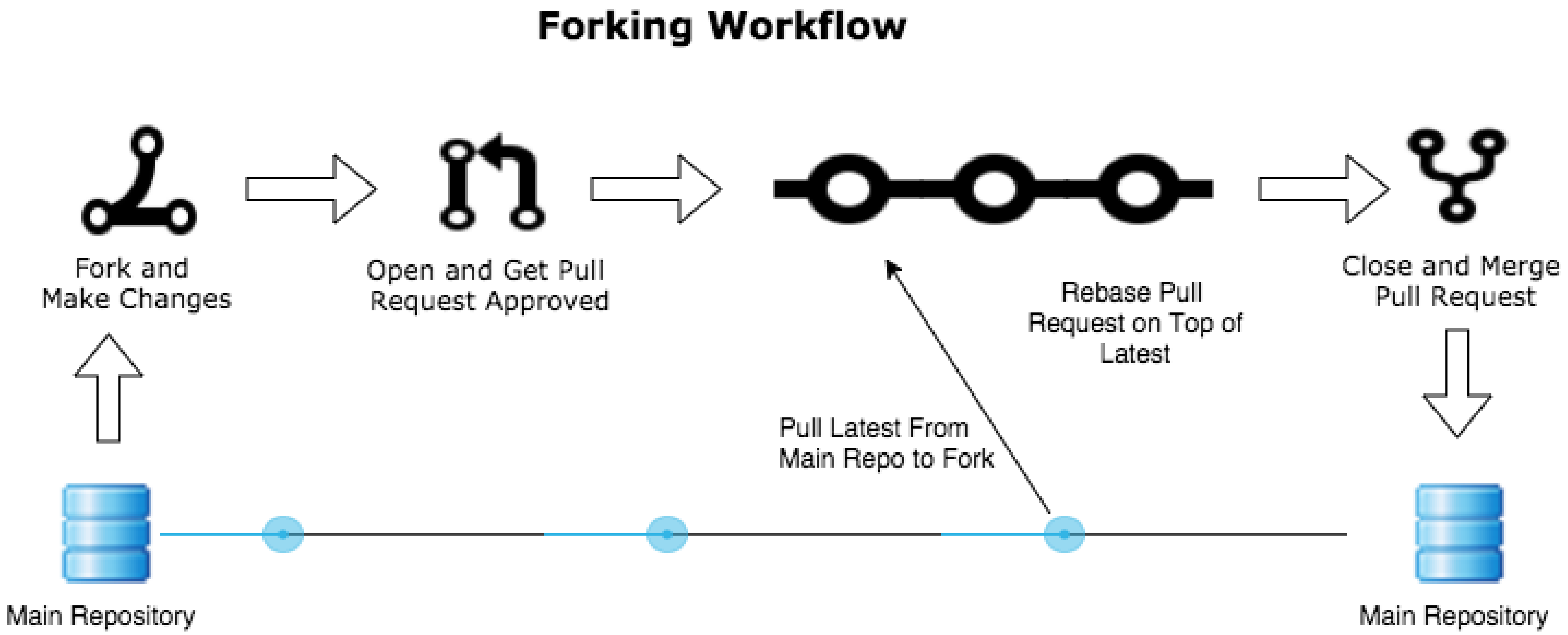

ADD COMMITS
DISCUSS AND REVIEW
CREATE A BRANCH
Create a branch in your project where you can safely experiment and make changes.
OPEN A PULL REQUEST
Use a pull request to get feedback on your changes from people down the hall or ten time zones away.
MERGE AND DEPLOY
Merge your changes into your master branch and deploy your code.
crunchify.com


Work requested
First commit
PR issued
PR picked up
PR merged
Work released
Start time
Fun time
Pick up time
Review time
Release time
Merge time
Cycle time
Lead time

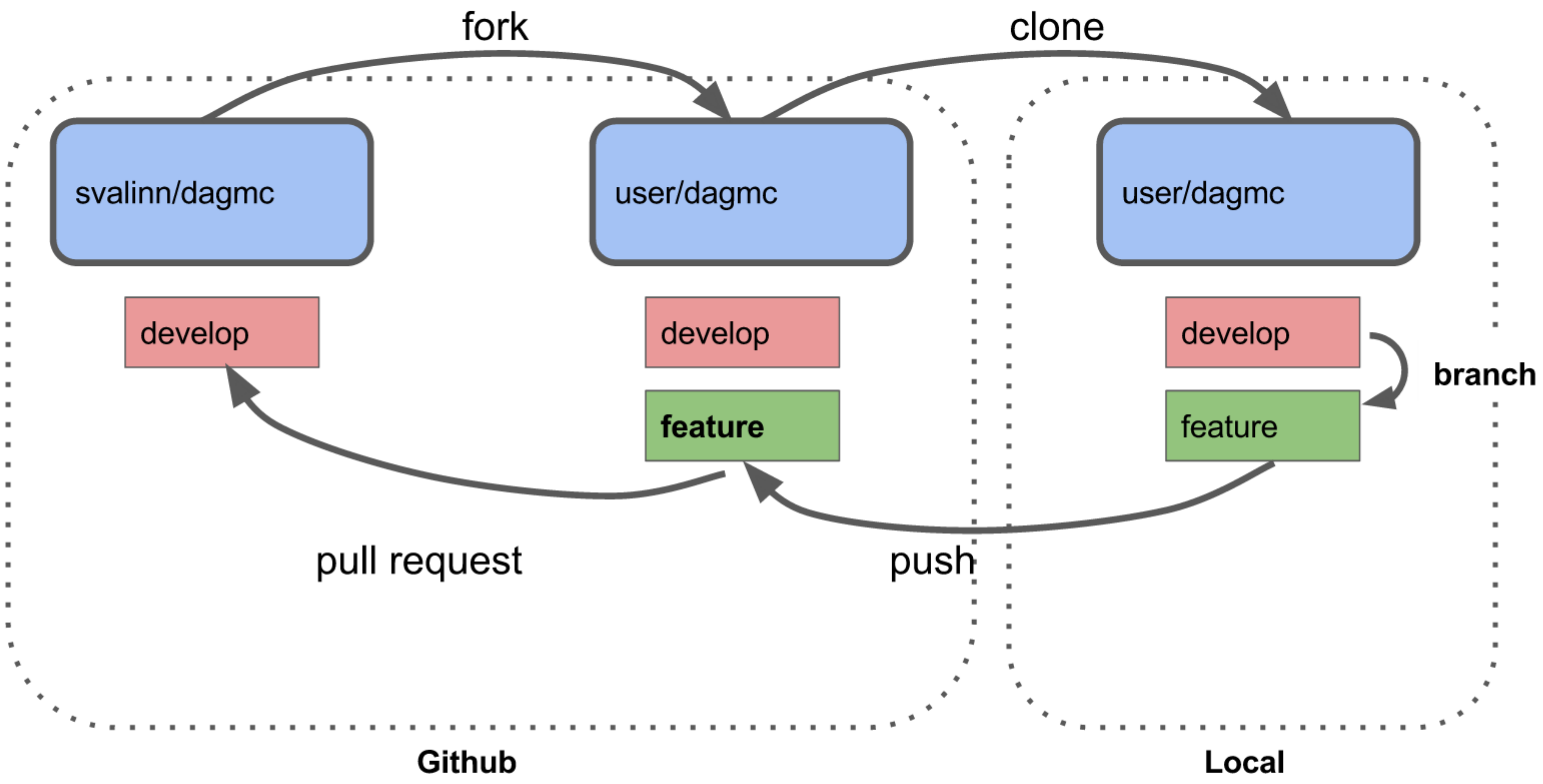
Git workflow: Fork

AKA **Git pull request workflow**

Working with Fork Workflow


Forking Workflow
Fork and Make Changes
Open and Get Pull Request Approved
Rebase Pull Request on Top of Latest
Close and Merge Pull Request
Pull Latest From Main Repo to Fork
Main Repository
Main Repository

Working with Fork Workflow



Git workflow: Working with Gitflow

Git workflow: Working with Gitflow

Gitflow is a Git workflow implementing tool that helps with continuous software development and implementing DevOps practices. It was first published and made popular by Vincent Driessen at nvie.

The Gitflow Workflow defines a strict branching model designed around the project release.

This provides a robust framework for managing larger projects.

Gitflow is ideally suited for projects that have a scheduled release cycle and for the DevOps best practice of continuous delivery.

Git workflow: Working with Gitflow

This workflow doesn’t add any new concepts or commands beyond what’s required for the Feature Branch Workflow.

Instead, it assigns very specific roles to different branches and defines how and when they should interact.

In addition to feature branches, it uses individual branches for preparing, maintaining, and recording releases.

Of course, you also get to leverage all the benefits of the Feature Branch Workflow: pull requests, isolated experiments, and more efficient collaboration.

Git workflow: Working with Gitflow

Gitflow is really just an abstract idea of a Git workflow. This means it dictates what kind of branches to set up and how to merge them together.

The git-flow toolset is an actual command line tool that has an installation process.

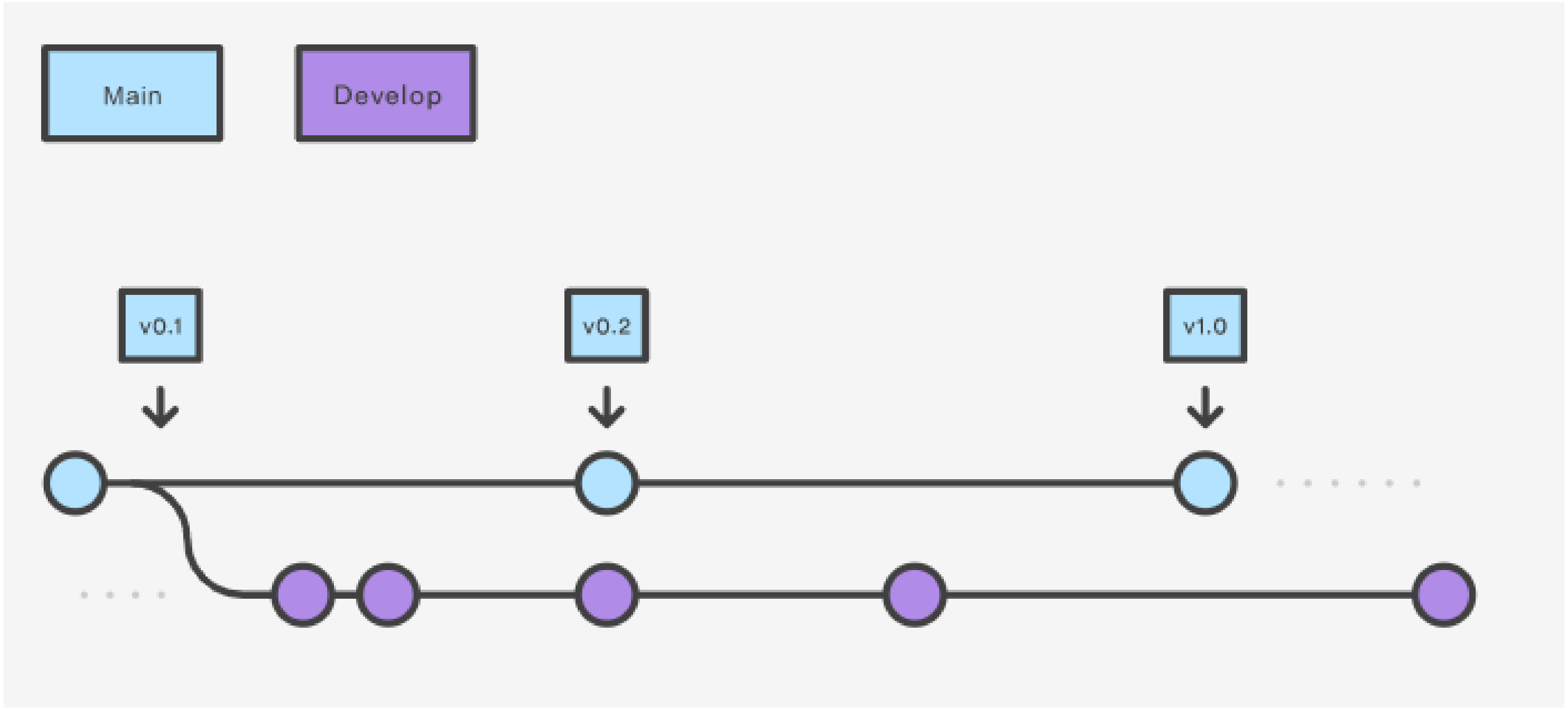
```
$ brew install git-flow
$ apt-get install git-flow
$ yum install git-flow
$ https://git-scm.com/download/win
```

After installing git-flow you can use it in your project by executing git flow init. Git-flow is a wrapper around Git.

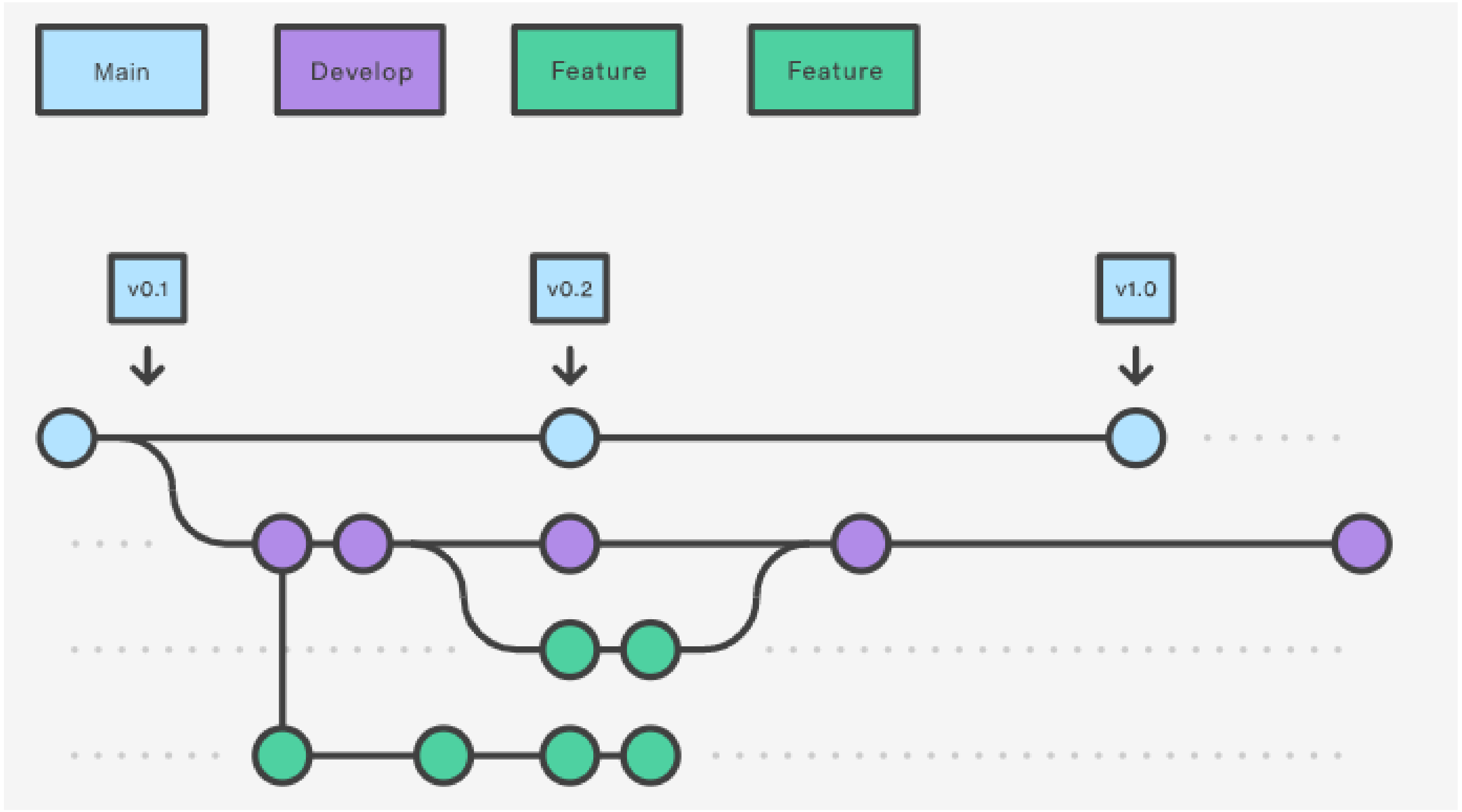
Types of branch in Gitflow

- Major release - main
- development - develop
- feature
- Minor release - release
- hotfix
- support

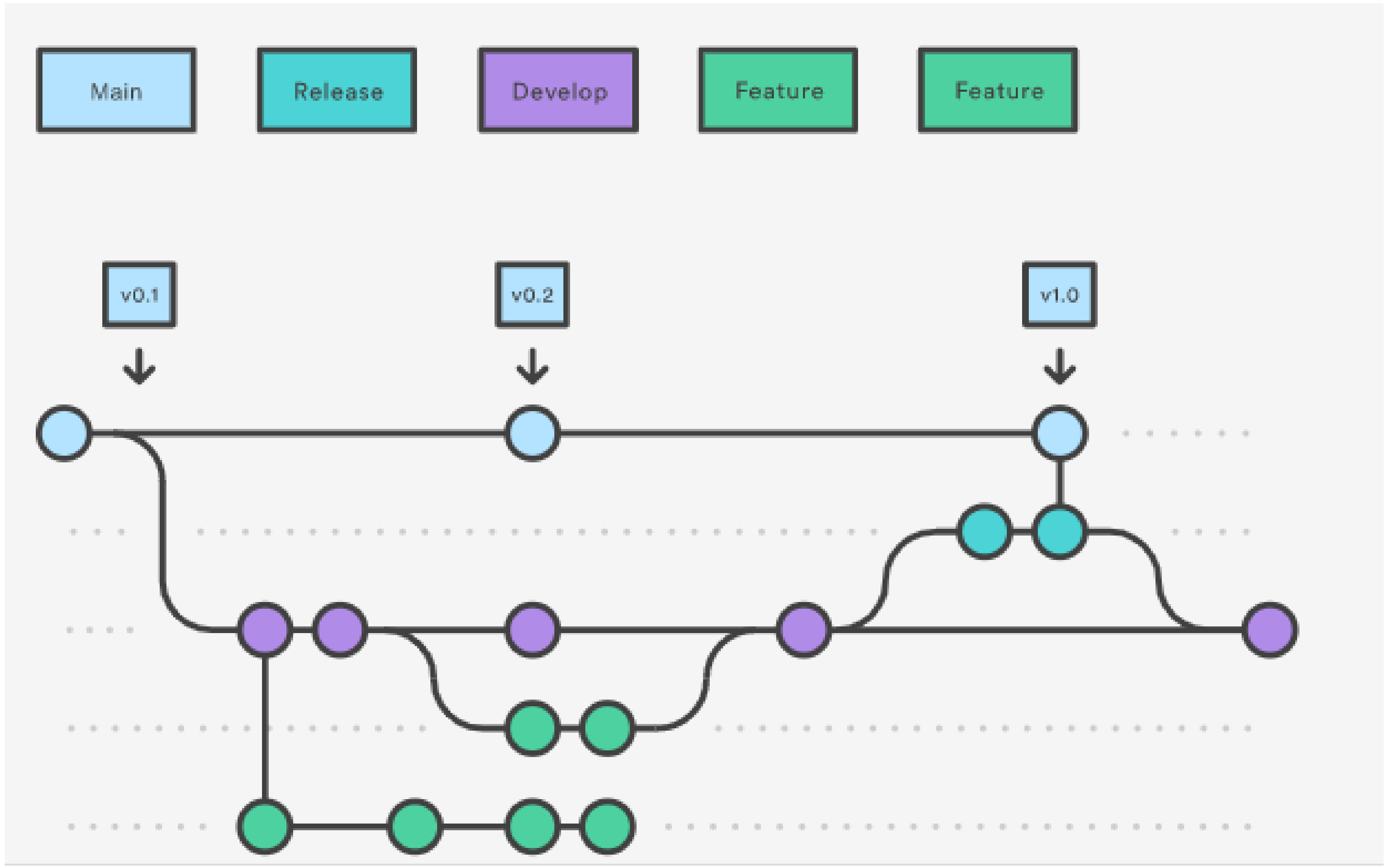
Working with Gitflow: Develop & Main Branches



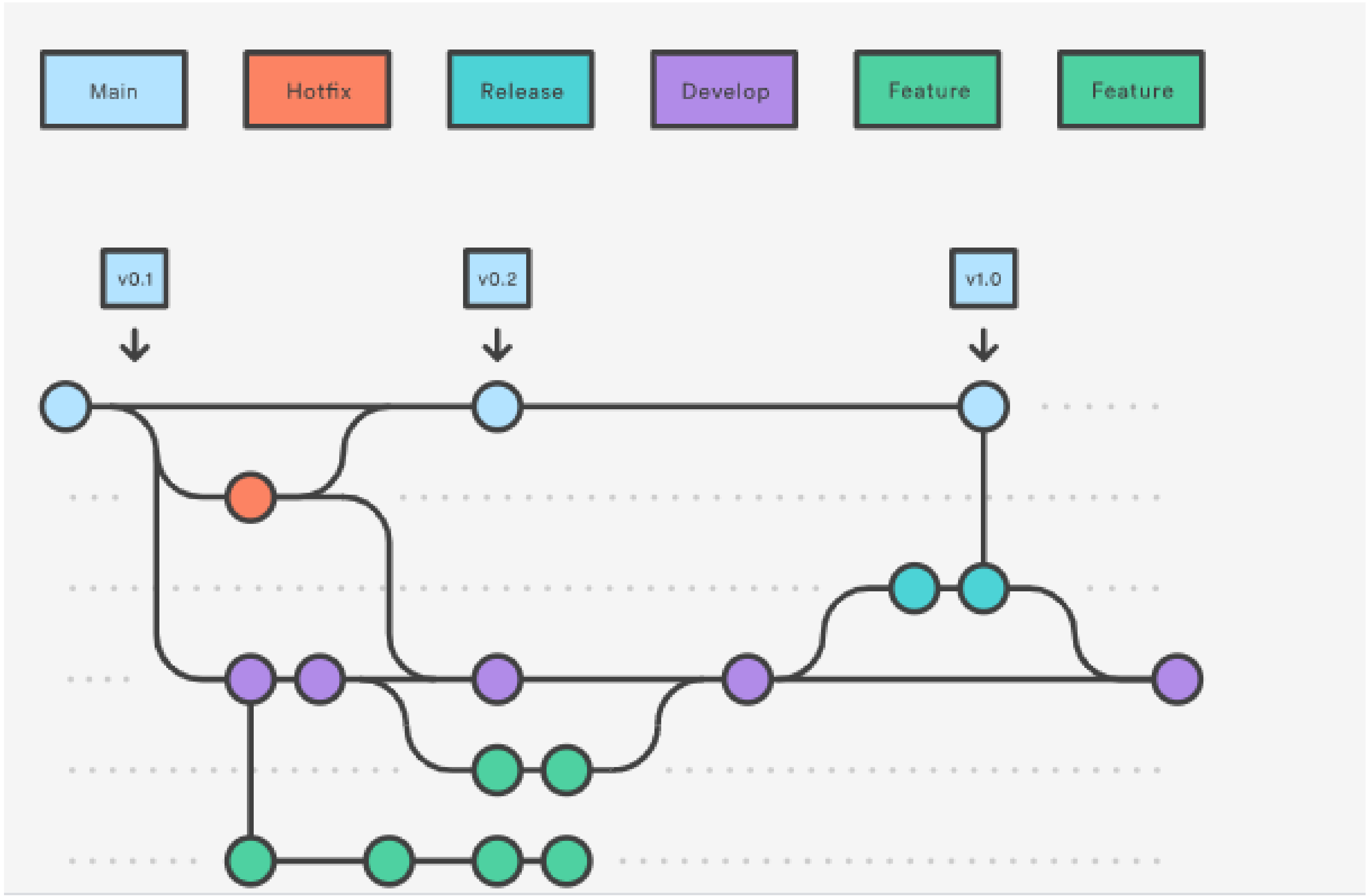
Working with Gitflow: Feature Branches



Working with Gitflow: Release Branches

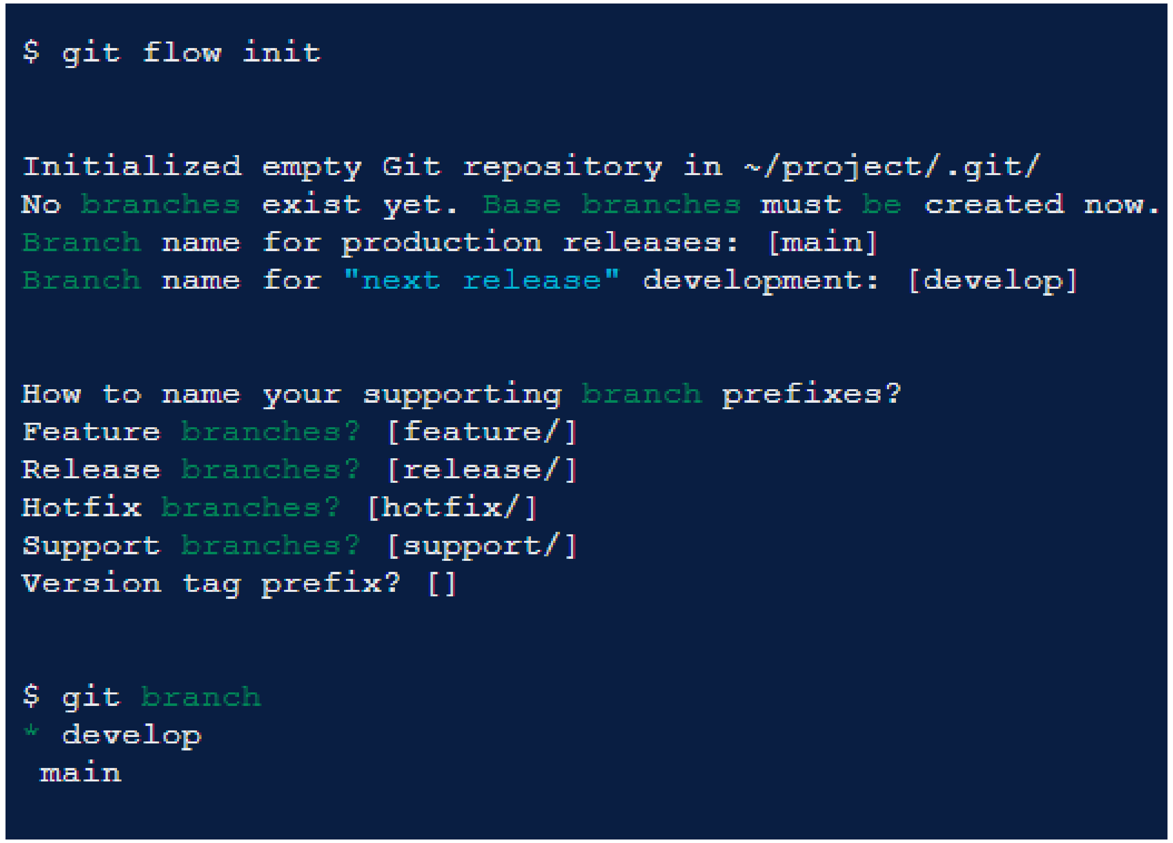


Working with Gitflow: Hotfix Branches



Git workflow: Working with Gitflow

The **git flow init** command is an extension of the default git init command and doesn't change anything in your repository other than creating branches for you.



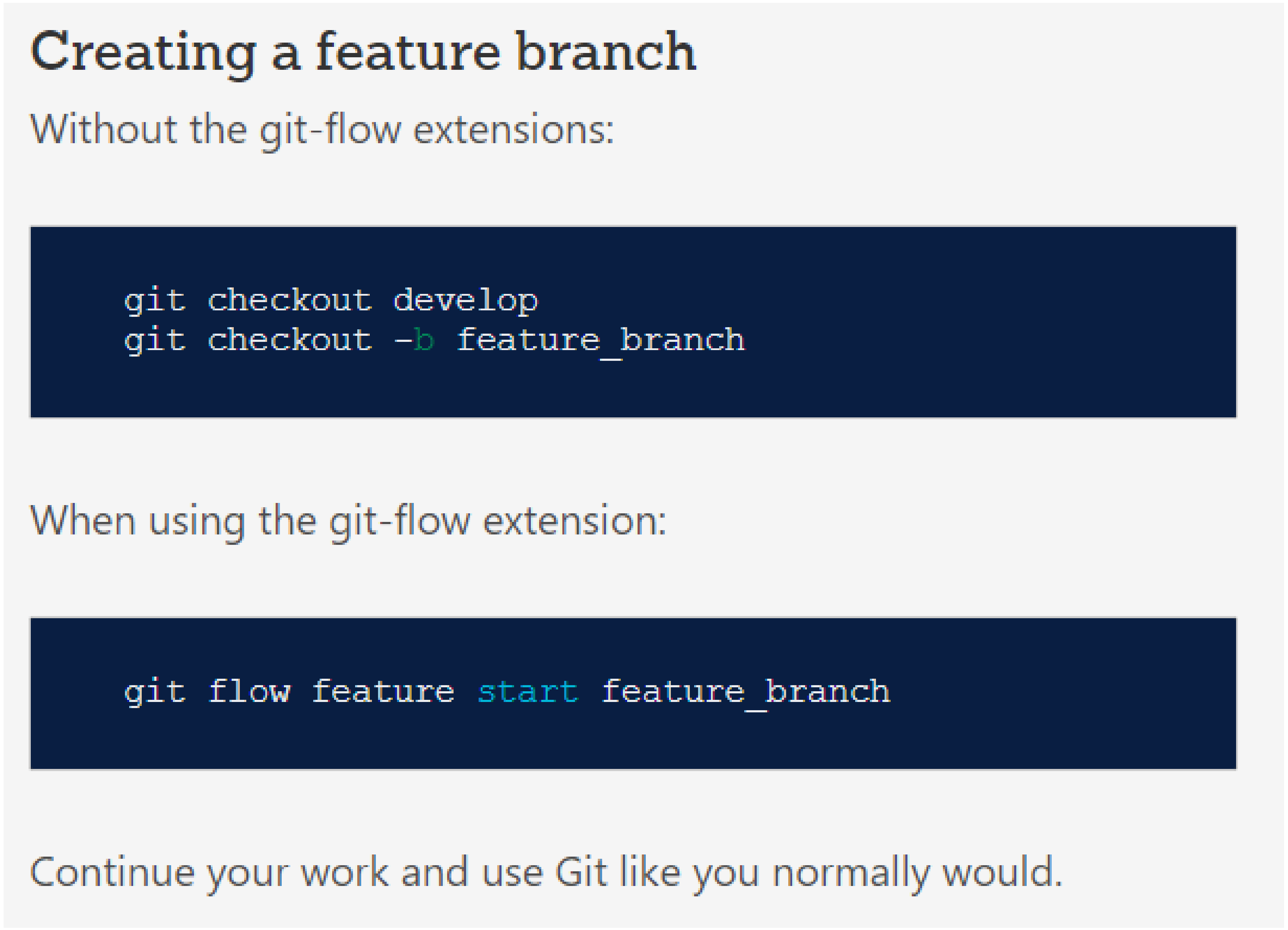
```
$ git flow init


Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [main]
Branch name for "next release" development: [develop]


How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []


$ git branch
* develop
 main
```

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow



Creating a feature branch

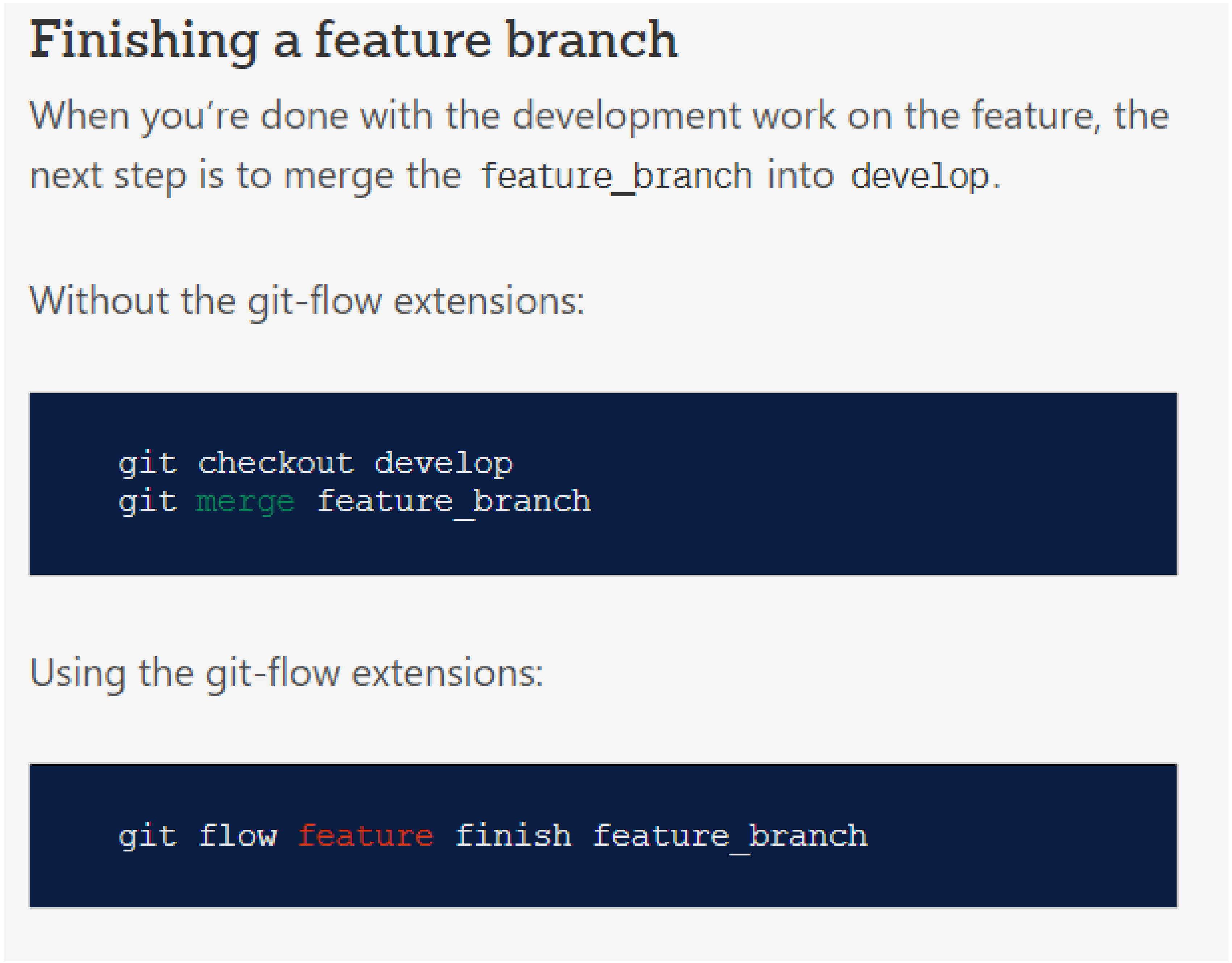
Without the git-flow extensions:

```
git checkout develop
git checkout -b feature_branch
```

When using the git-flow extension:

```
git flow feature start feature_branch
```

Continue your work and use Git like you normally would.



Finishing a feature branch

When you're done with the development work on the feature, the next step is to merge the feature_branch into develop.

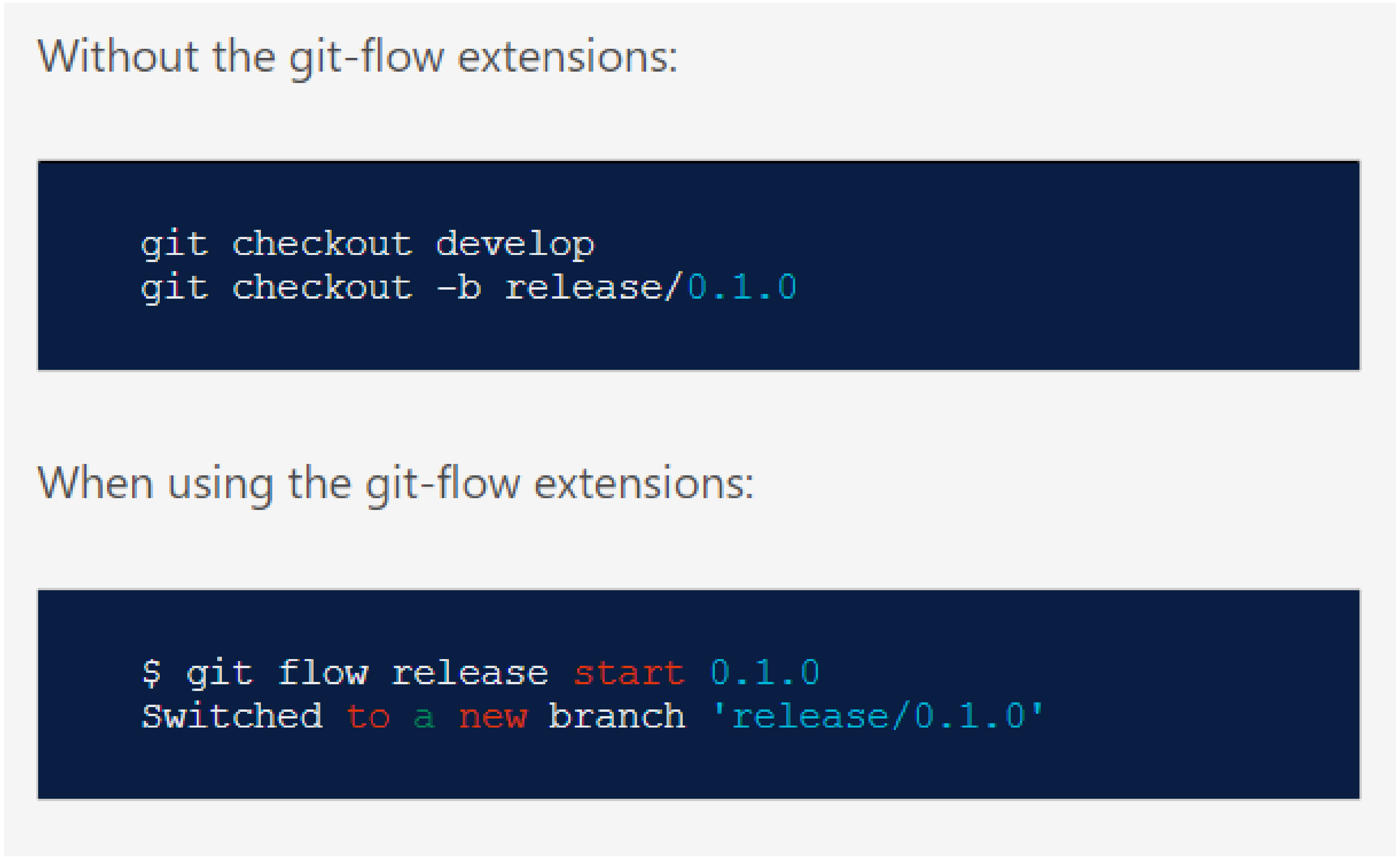
Without the git-flow extensions:

```
git checkout develop
git merge feature_branch
```

Using the git-flow extensions:

```
git flow feature finish feature_branch
```

Release Branches



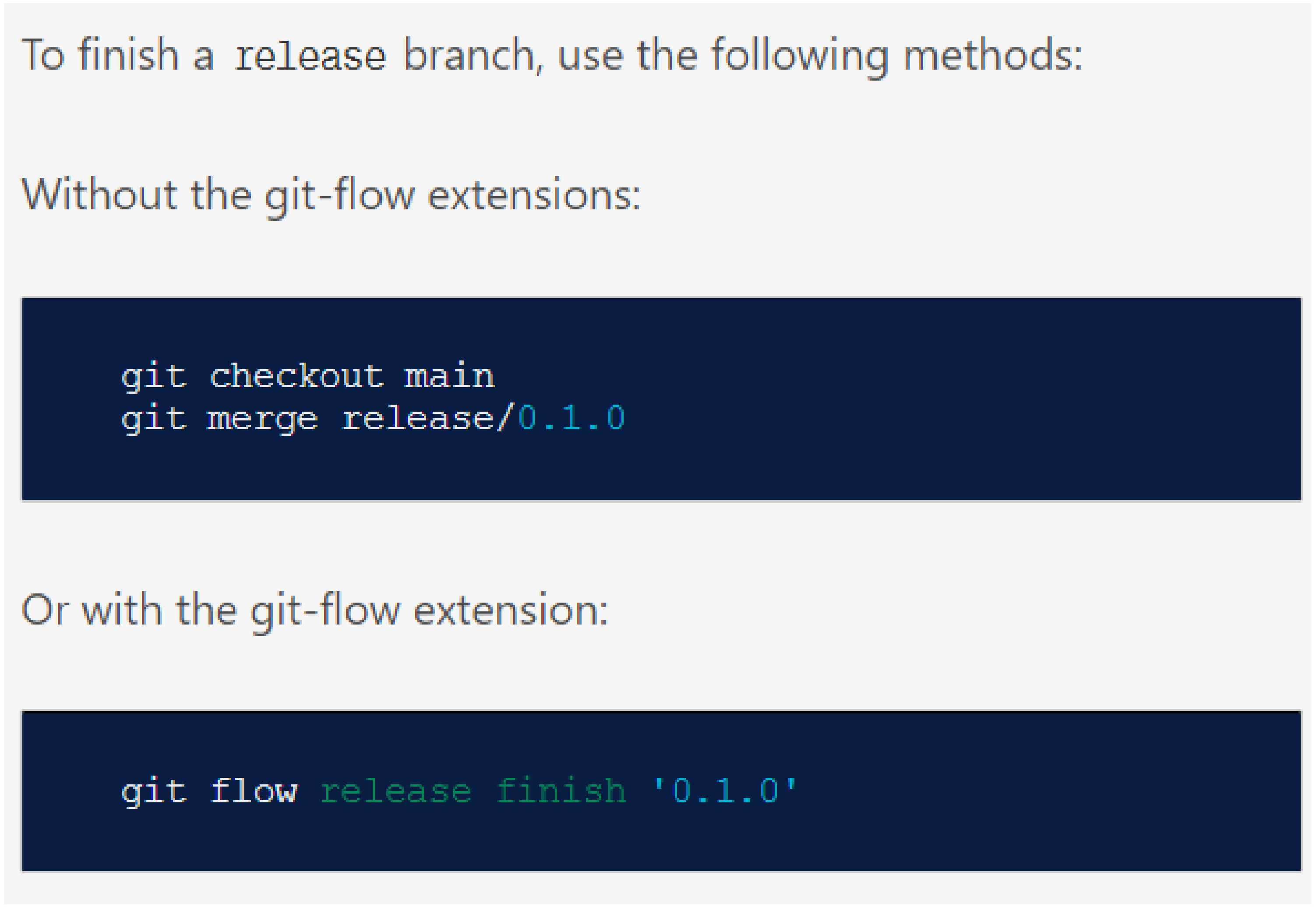
Without the git-flow extensions:

```
git checkout develop
git checkout -b release/0.1.0
```

When using the git-flow extensions:

```
$ git flow release start 0.1.0
Switched to a new branch 'release/0.1.0'
```

Release Branches: Finish



To finish a `release` branch, use the following methods:

Without the git-flow extensions:

```
git checkout main
git merge release/0.1.0
```

Or with the git-flow extension:

```
git flow release finish '0.1.0'
```