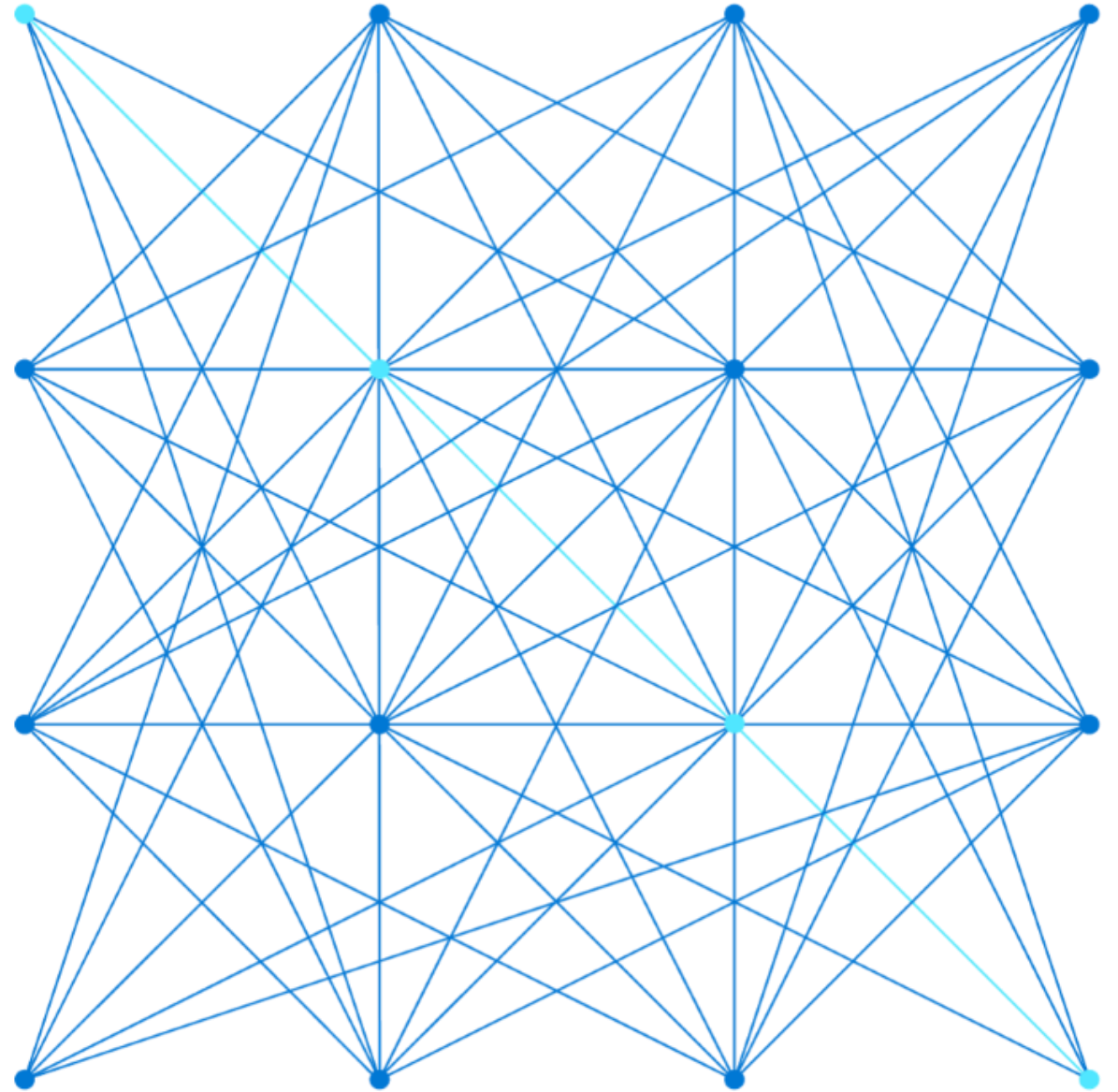


AZ-400.00

Module 4:

Working with Git for Enterprise DevOps



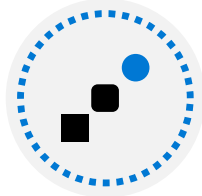
Lesson 01: Module overview



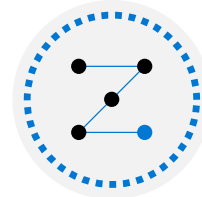
Module overview



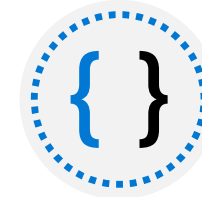
Lesson 1: Module overview



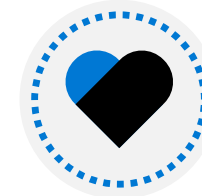
Lesson 2: How to structure your Git Repo



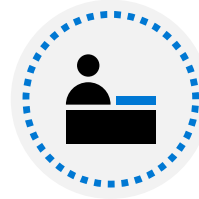
Lesson 3: Git branching workflows



Lesson 4: Collaborating with pull requests in Azure Repos



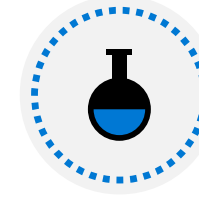
Lesson 5: Why care about Git Hooks ?



Lesson 6: Fostering inner source



Lesson 7: Managing Git Repositories



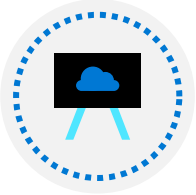
Lesson 8: Lab



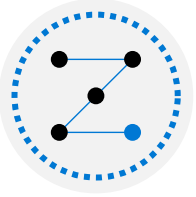
Lesson 9: Module review and takeaways

Learning objectives

After completing this module, students will be able to:



Explain how to structure Git Repos



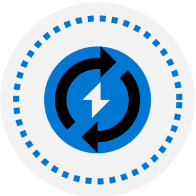
Describe Git branching workflows



Leverage pull requests for collaboration and code reviews

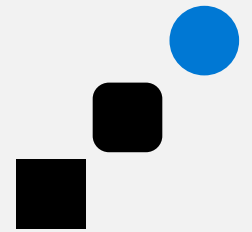


Leverage Git hooks for automation



Use Git to foster inner source across the organization

Lesson 02: How to structure your Git Repo



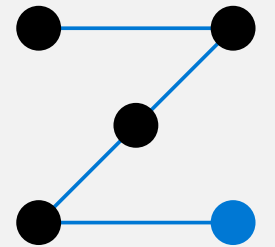
(#mw) A **repository** is a place where the **history** of your work is stored.

Two approaches:

1. '**Monorepo**' (one big repo with everything in it)
 - Simple. Everything in one place.
 - Anybody can fix bugs anywhere in the repo
 - Can become unwieldy.
 - Can lead to complex, tightly coupled dependency graphs.
2. '**Multiple repos**' (many small repo's each with it's own concern, e.g WFE/AppServices/DataServices)
 - Clear separation of concerns
 - Loose coupling.
 - Developers can download only the repo they need.
 - Can be harder to understand dependencies between repos?
 - Can result in different teams using different techniques and development approaches.
 - Could slow development down as people wait for bug fixes to be published by other teams.

Azure DevOps projects can contain multiple repositories, often 1 per solution.

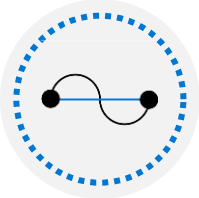
Lesson 03: Git branching workflows



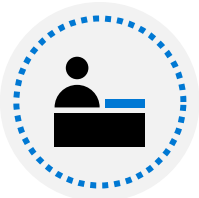
Branching workflow types



Feature branching – All feature development should take place in a dedicated branch instead of the master branch.



GitFlow branching – A strict branching model designed around the project release. (#mw: [link](#))



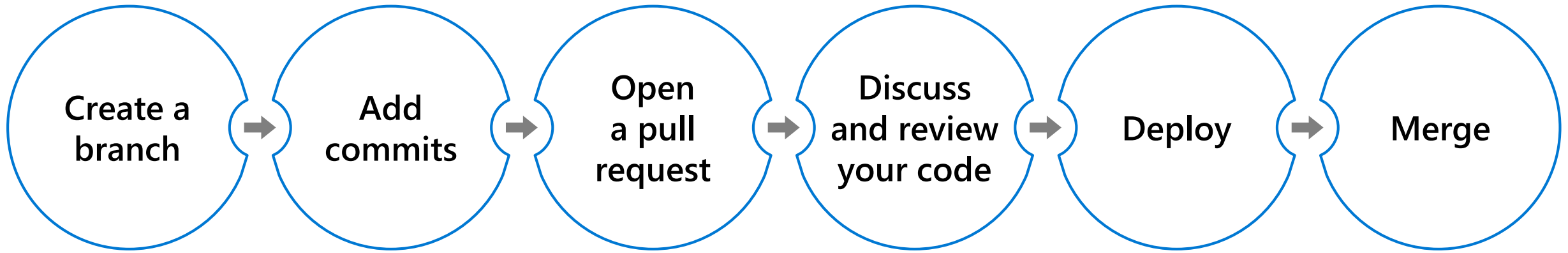
Forking Workflow – Every developer uses a server-side repository. (#mw: [link](#))



Evaluate the workflow:

- Does this workflow scale with team size?
- Is it easy to undo mistakes and errors with this workflow?
- Does this workflow impose any new unnecessary cognitive overhead to the team?

Feature branch workflow

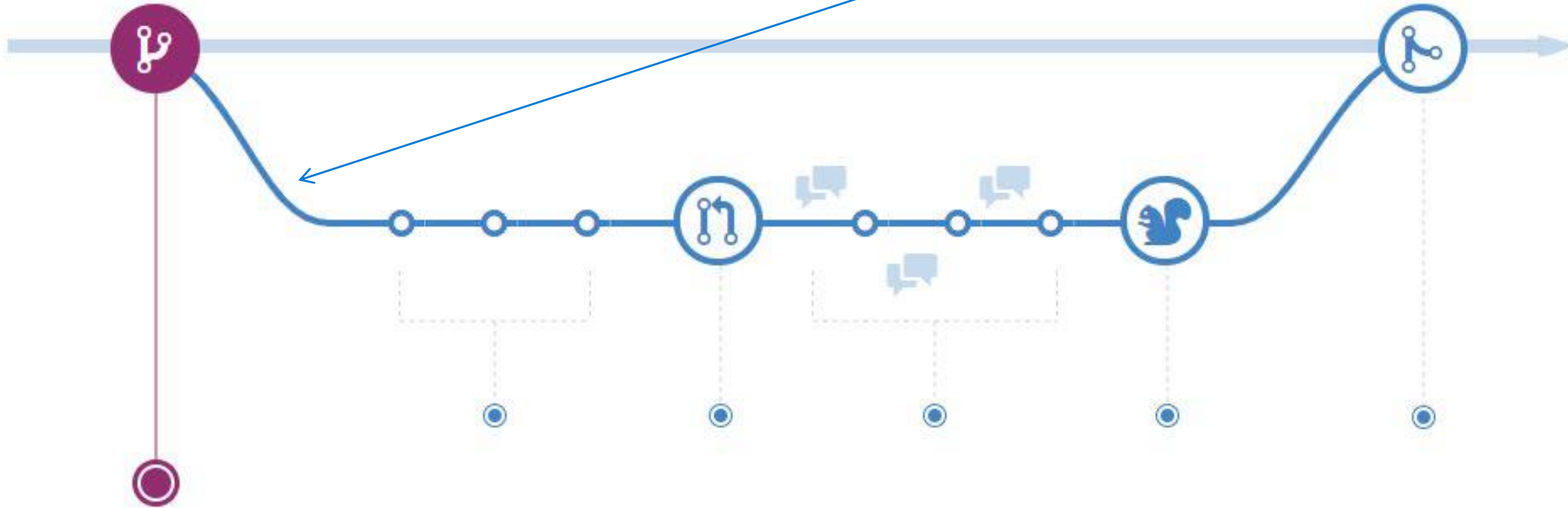


All feature development should take place in a dedicated branch instead of the master branch

Encapsulating feature development leverages pull requests, which are a way to initiate discussions around a branch.

Share a feature with others without touching any official code.

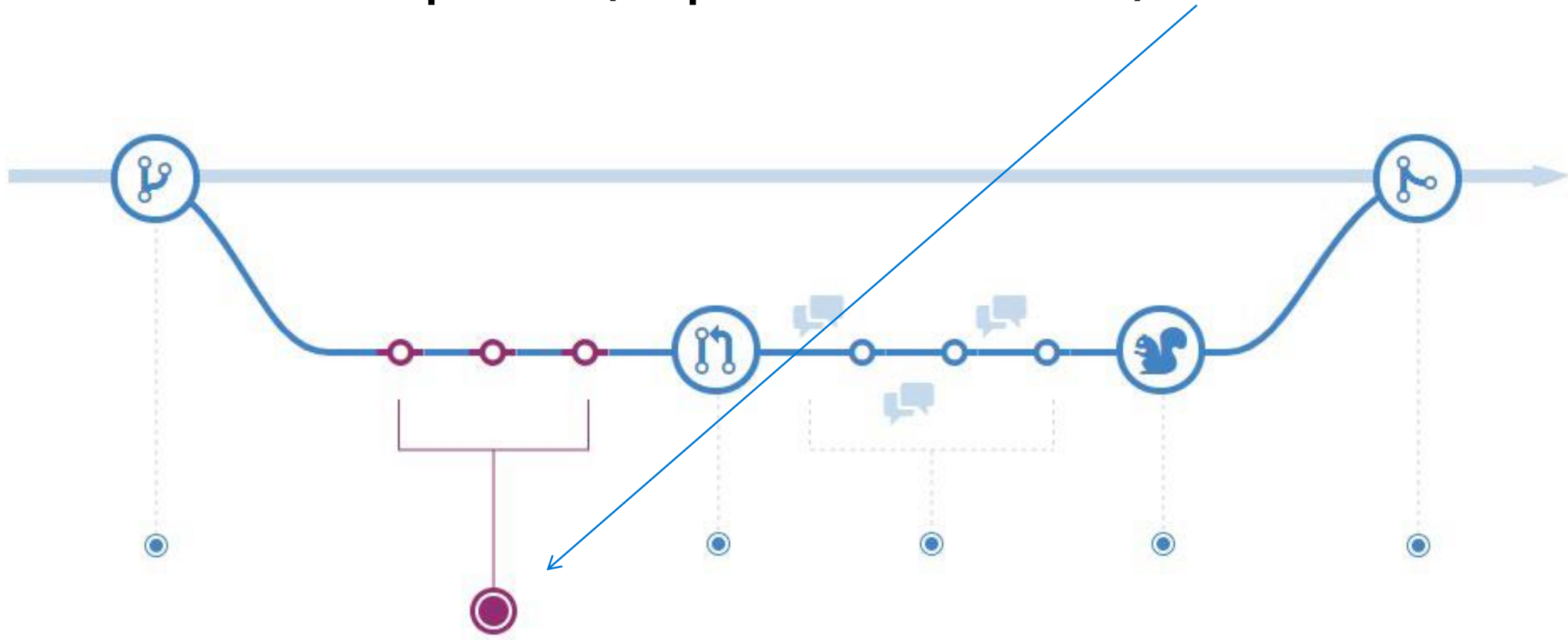
Git Workflow development (step 1 – create a branch)



Notes:

- Core Rule=Everything on the 'trunk' (i.e. master branch) must always be deployable

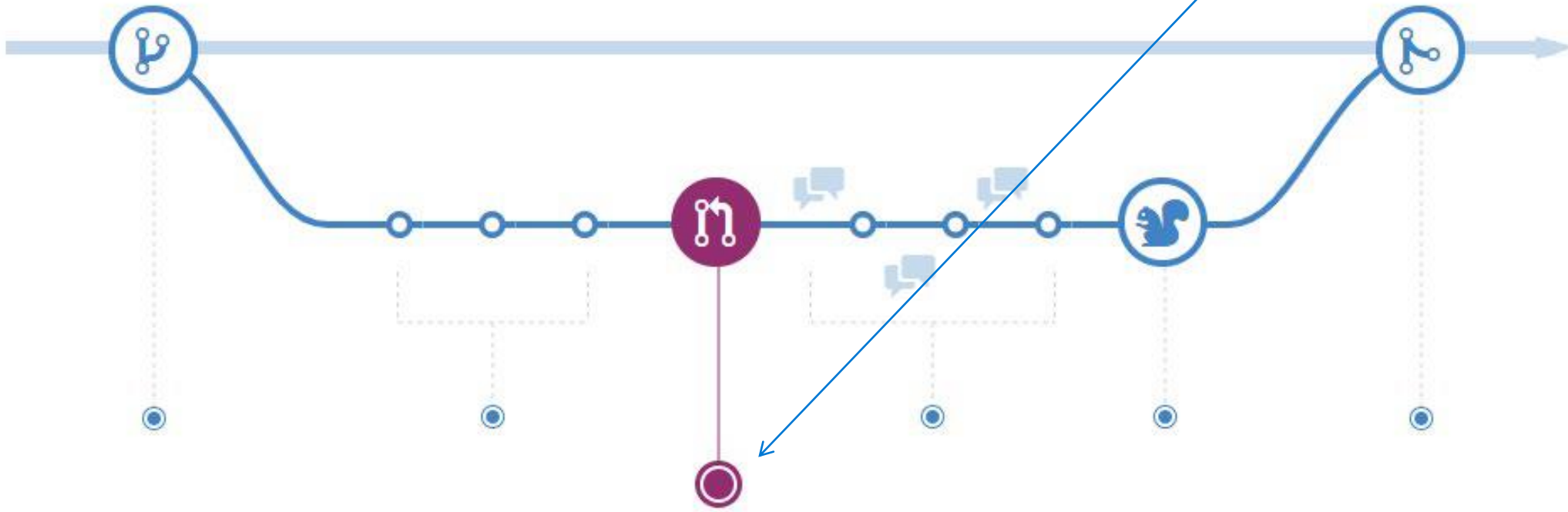
Git Workflow development (step 2 – add commits)



Notes:

- Commit early and commit often

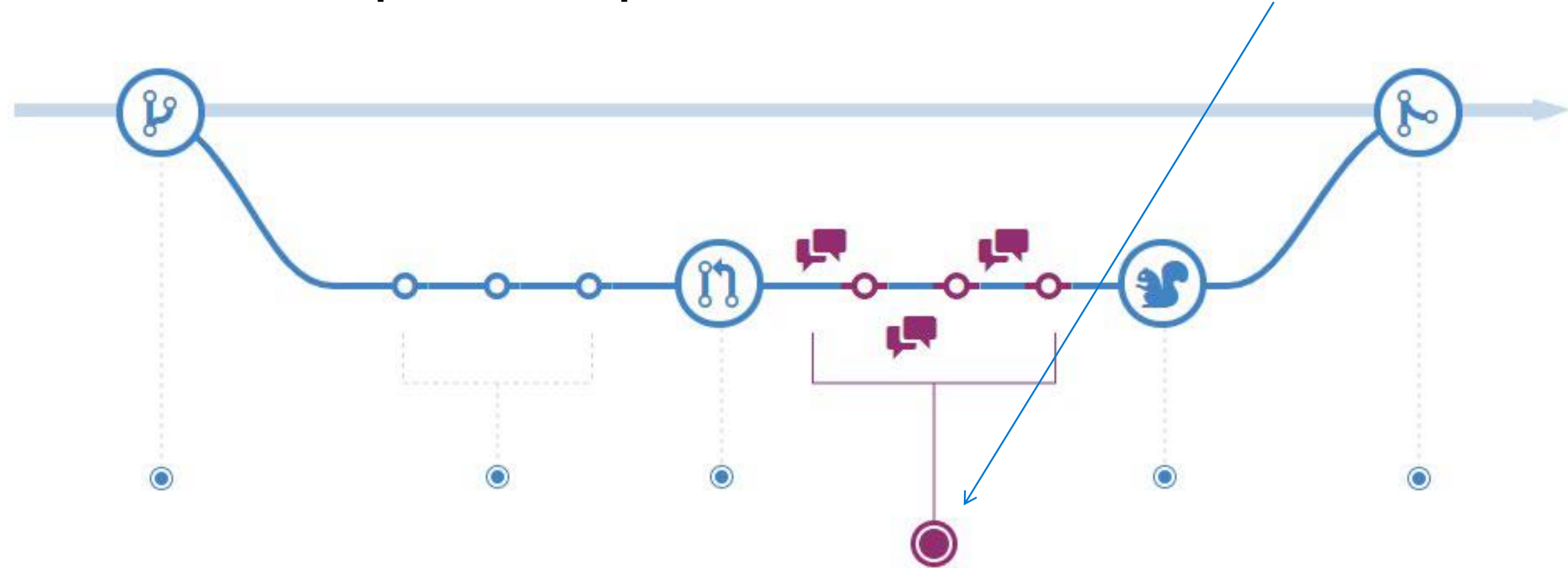
Git Workflow development (step 3 – open a pull request)



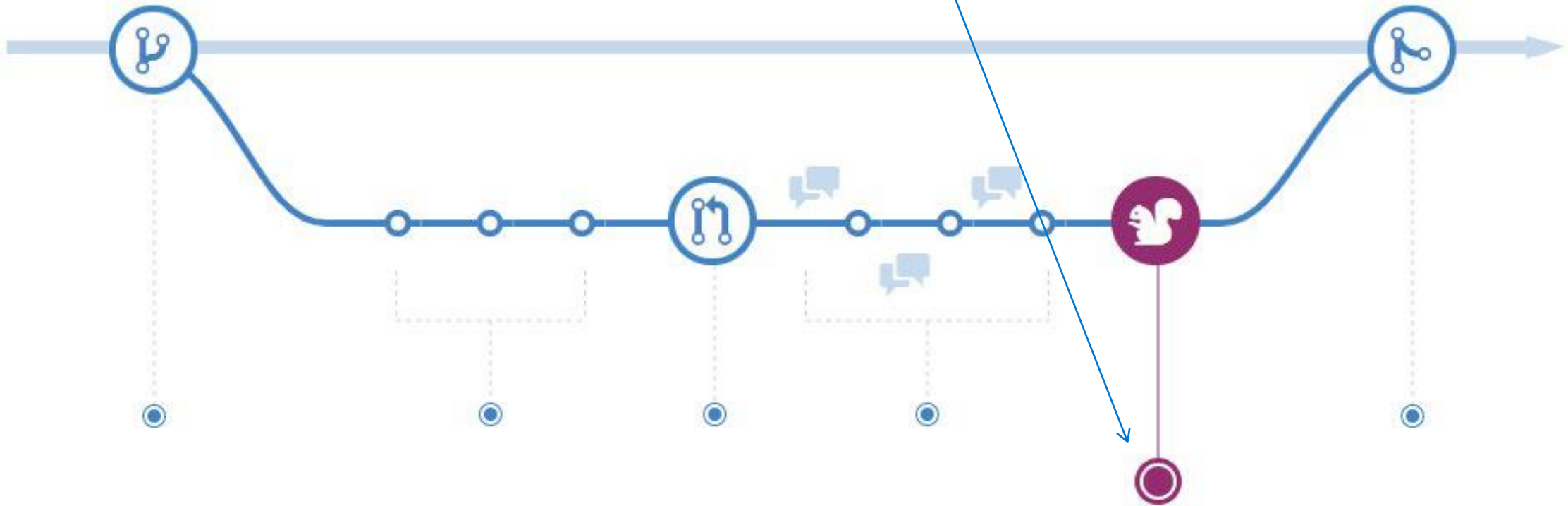
Notes:

- To initiate discussion or ask for help with regard to developing the feature

Git Workflow development (step 4 – discuss and review code)



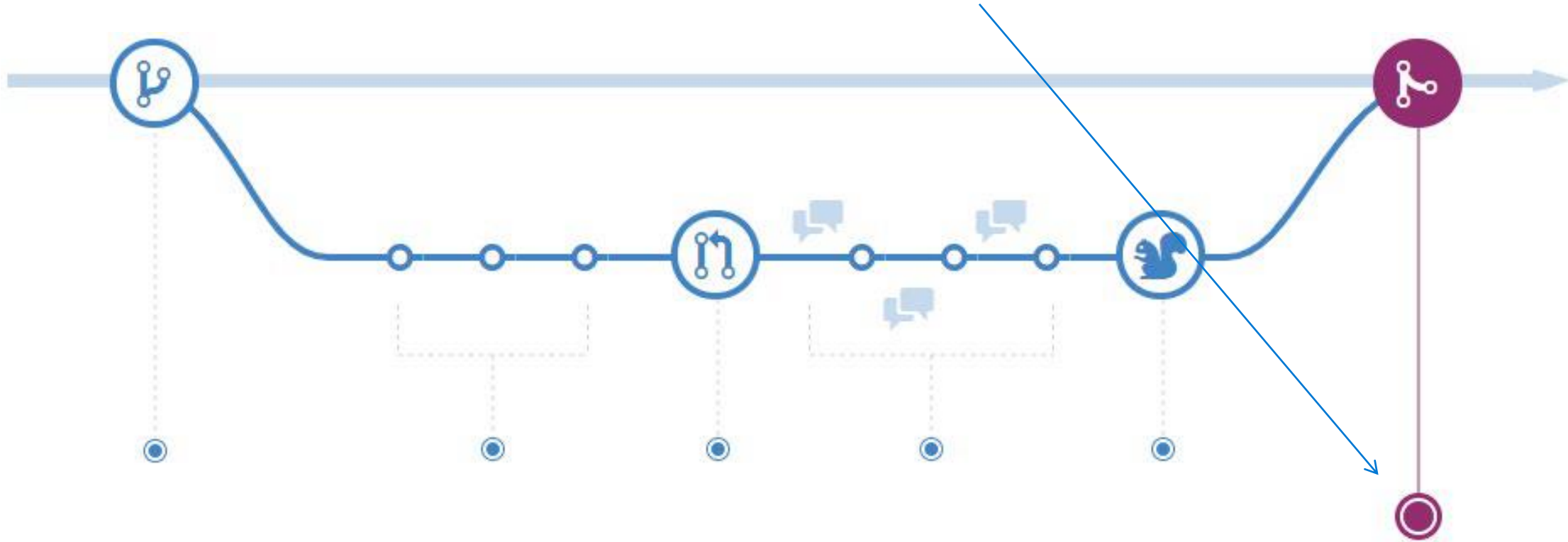
Git Workflow development (step 5 – deploy)



Notes:

- With Git, you can deploy from a branch for final testing in an environment before merging to master.

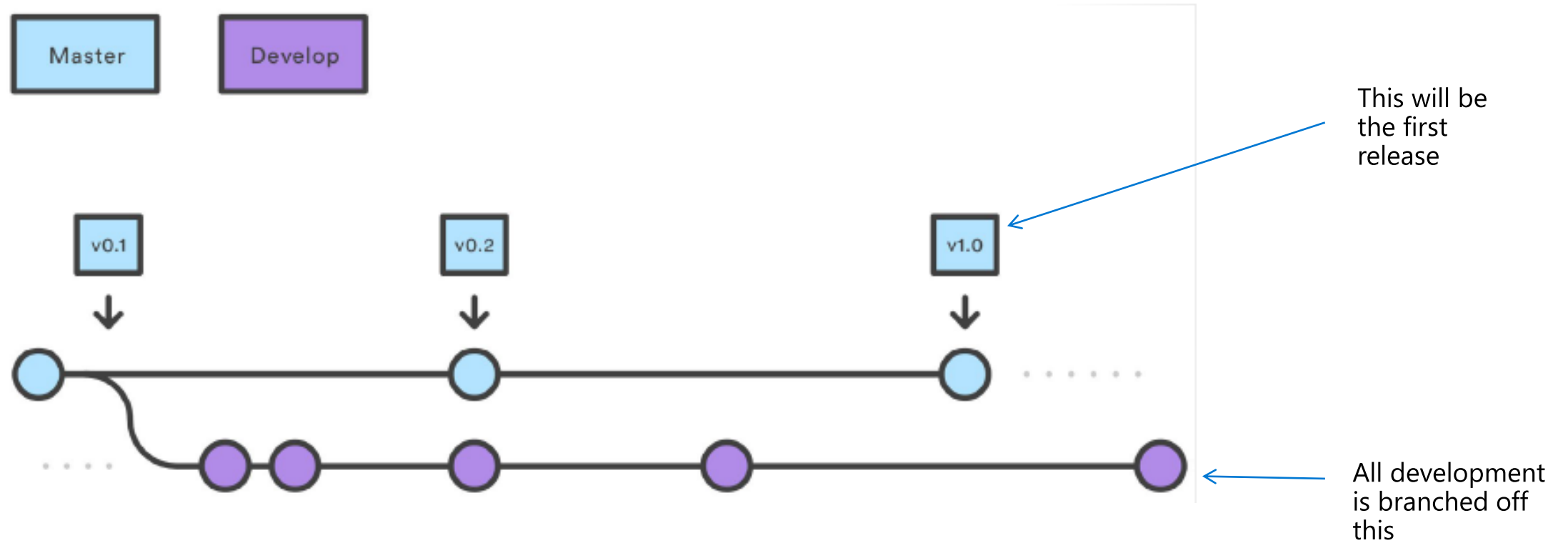
Git Workflow development (step 6 – merge to master)



Notes:

- Pull Requests preserve a record of the historical changes to your code.

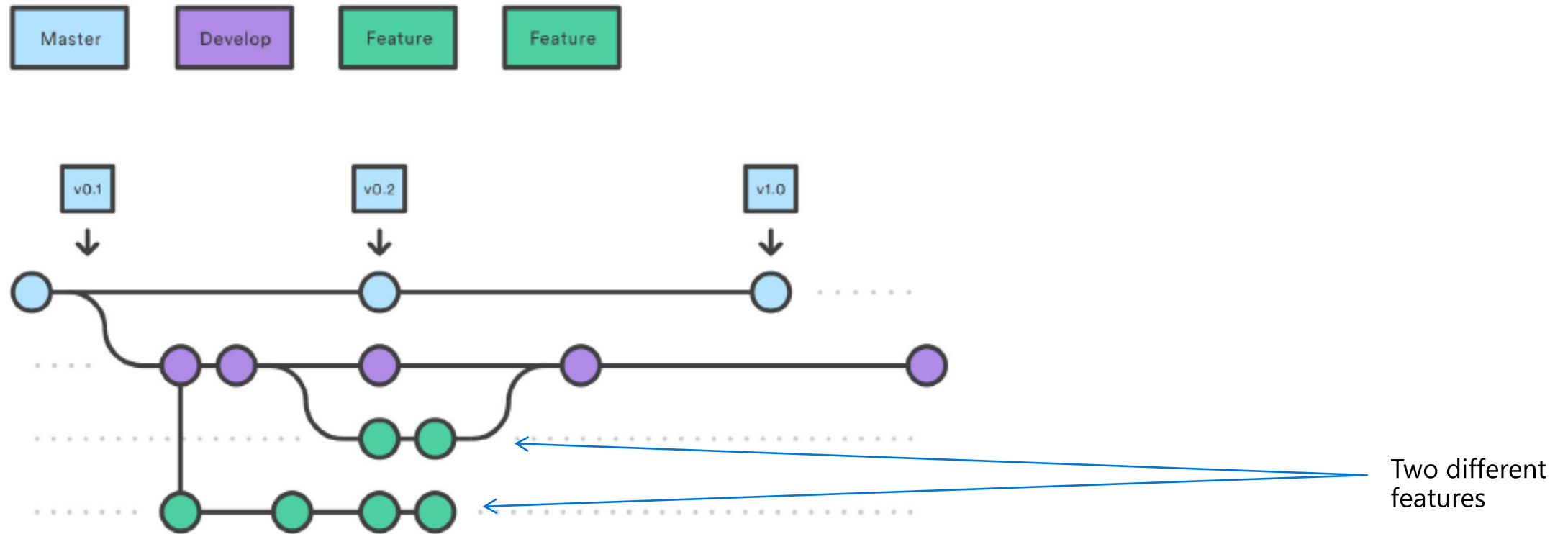
GitFlow branch workflow (step 1 – Initial setup)



Notes:

- [Gitflow](#) is a layer on top of Git workflow, i.e. it is an abstraction
- GitFlow is a separate install that simplifies the commands used for feature branch flow.

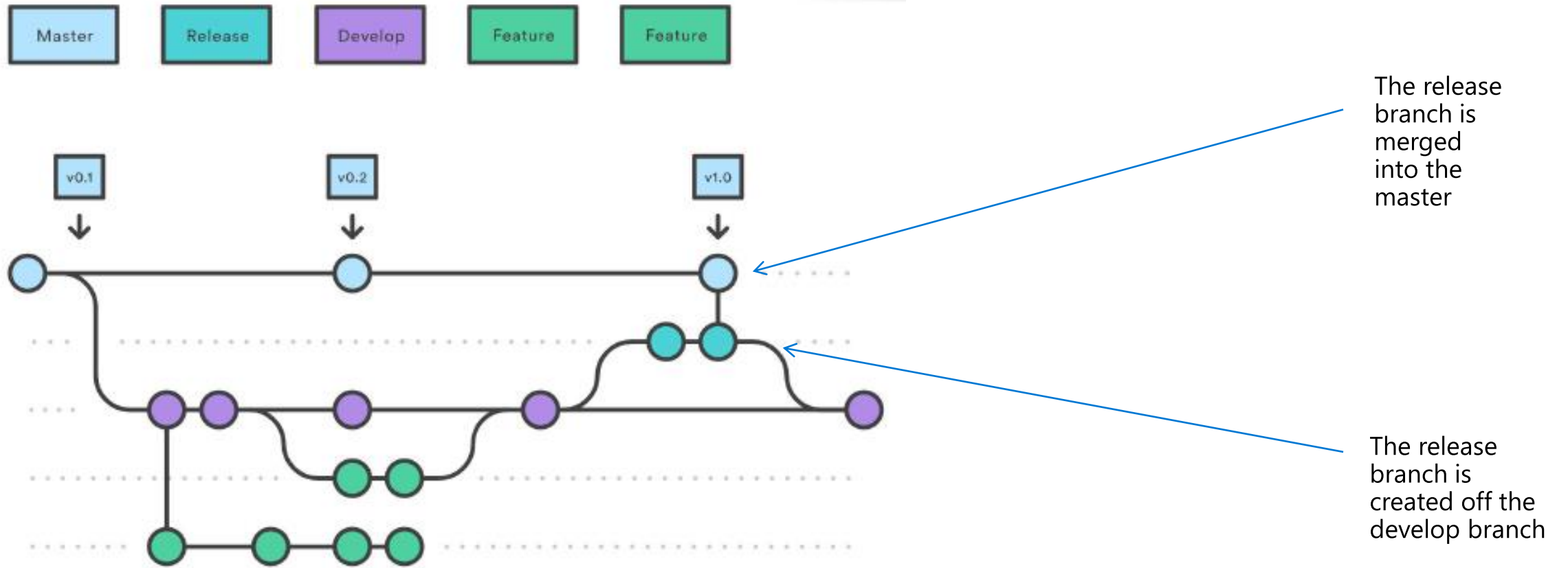
GitFlow branch workflow (step 2 – Feature Branches)



Notes:

- Features are branched off the Develop branch

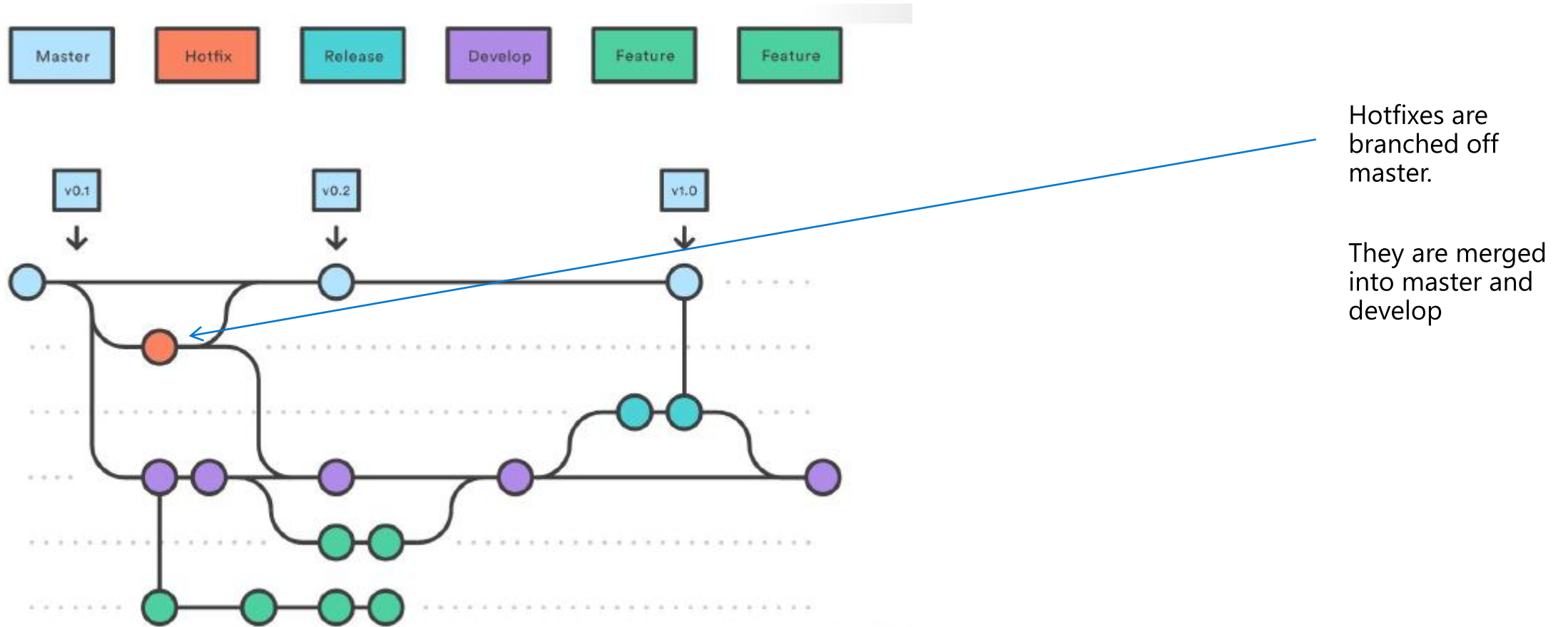
GitFlow branch workflow (step 3 – release branches)



Notes:

- Using a dedicated branch to prepare releases makes it possible for one team to publish the current release while another team continues working on features for the next release

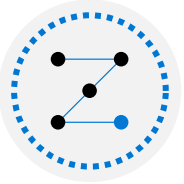
GitFlow branch workflow (step 4 – hotfixes)



Notes:

- Using a dedicated branch to prepare releases makes it possible for one team to publish the current release while another team continues working on features for the next release

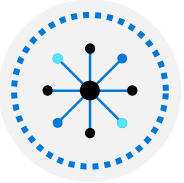
Forking workflow



Forking workflow gives every developer their own server-side repository



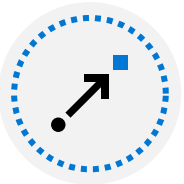
Each contributor has not one, but two Git repositories: a private local one and a public server-side one



Most often seen in public open-source projects



Contributions can be integrated without the need for everybody to push to a single central repository

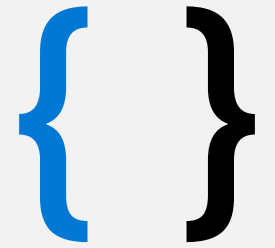


Typically follows a branching model based on the GitFlow workflow



Forked repositories use the standard git **clone** command

Lesson 04: Collaborating with pull requests



Collaborating with pull requests



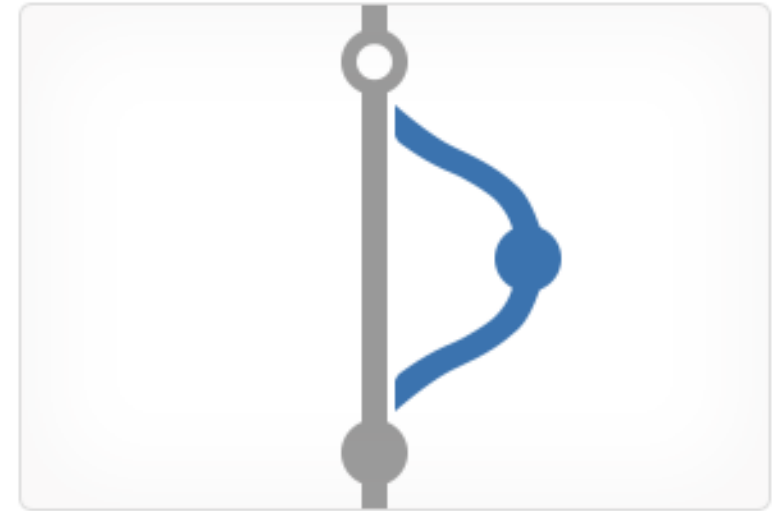
Branch

Develop features on a branch and create a pull request to get changes reviewed.



Discuss

Discuss and approve code changes related to the pull request.



Merge

Merge the branch with the click of a button.

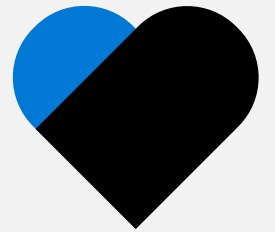
Pull requests let you tell others about changes.

Collaboration using the Shared Repository Model

Review and merge your code in a single collaborative process.

Be sure to provide good feedback and protect branches with policies.

Lesson 05: Why care about Git hooks ?

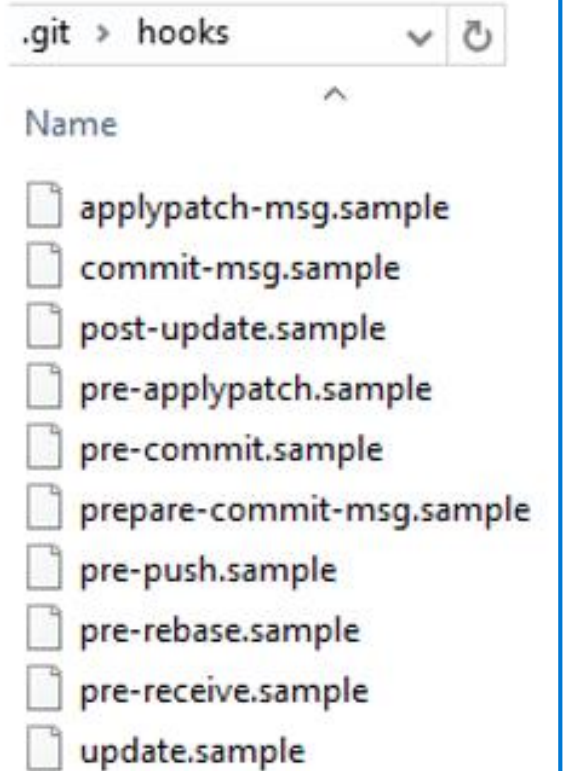


Why care about Git hooks?

A mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur

Use Git hooks to enforce policies, ensure consistency, and control your environment

Can be either client-side or server-side



Git hooks in action

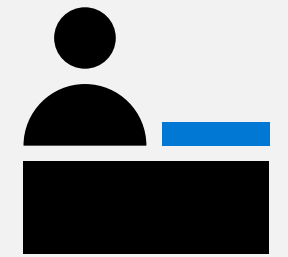
Will my code:

- Break other code?
- Introduce code quality issues?
- Take on a new dependency?

Some examples of how a github hooks might be used:

- In Enforcing preconditions for merging
- Verifying work Item Id association in your commit message
- Preventing you & your team from committing faulty code
- Sending notifications to your team's chat room (Teams, Slack, HipChat, etc)

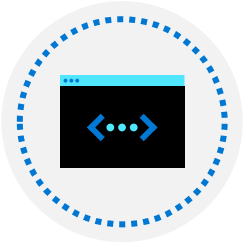
Lesson 06: Fostering inner source



Inner Source?????



AKA '*Internal Open Source*'.



Why?

- Allows everybody in the organisation to see all code.
- Developers can contribute to other projects by [forking](#).
- Microsoft uses this model with its source code.

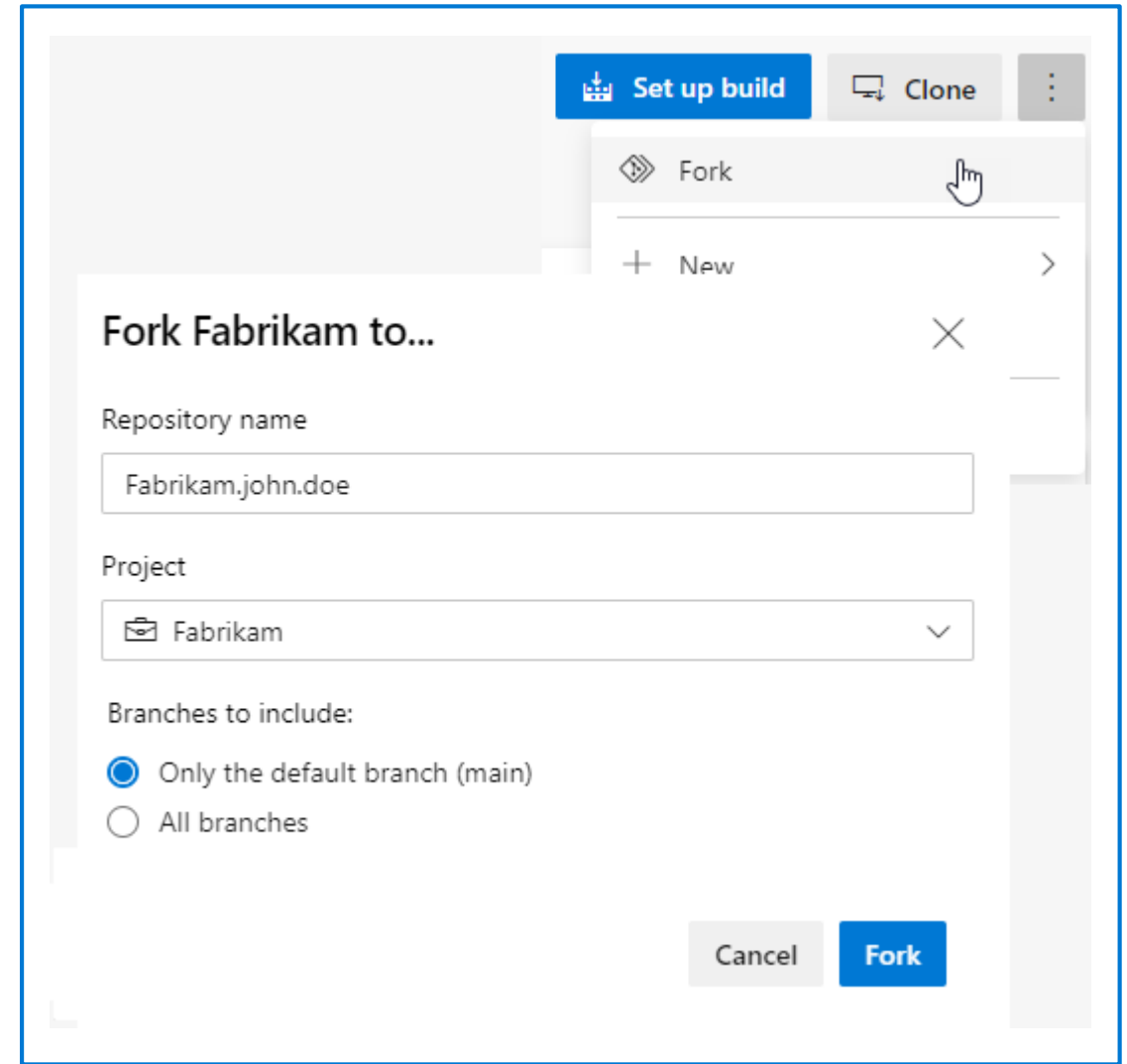
Implementing the fork workflow

What's in a [fork](#)?

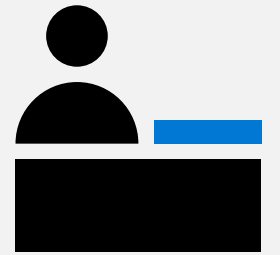
- Developer has a *Server-side* **copy** of a repo.

The forking workflow:

- Create a fork
- Clone it locally
- Make your changes locally and push them to a branch
- Create and complete a PR to upstream
- Sync your fork to the latest from upstream



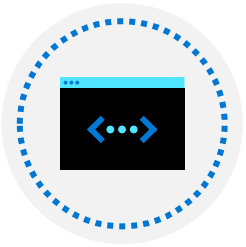
Lesson 07: Managing Git Repositories



Working with large repositories

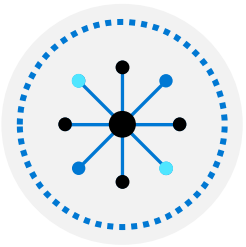


Common causes of large repos: long history and large binary files



Long history: use shallow clones

```
git clone -depth [depth] [clone-url]
```



Large binary files: use Git Virtual File System (GVFS)

(#mw) Now called [VFS for Git](#)

Purging repository data

Might need to delete files from repository:

- Reduce repository size
- Remove accidentally committed large file
- Remove committed file with sensitive data (passwords, keys)

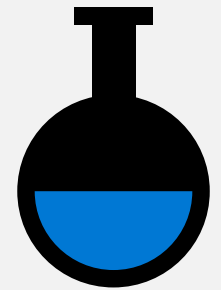
Use:

- [filter-branch command](#)
- [BFG Repo-Cleaner](#)

```
$ bfg --delete-files file_I_should_not_have_committed
```

```
$ bfg --replace-text passwords.txt
```

Lesson 08: Lab



Version controlling with Git in Azure Repos



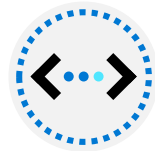
Lab overview:

In this lab, you will learn how to work with branches and repositories in Azure DevOps.

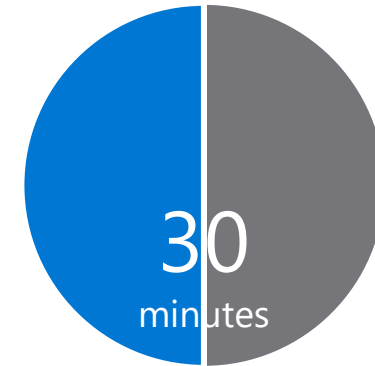
Objectives:

Work with branches in Azure Repos

Work with repositories in Azure Repos



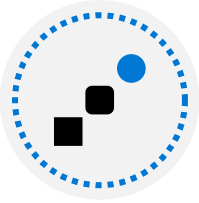
Duration:



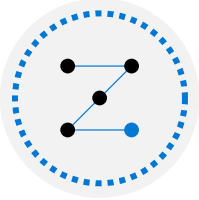
Lesson 09: Module review and takeaways



What did you learn?



Explain how to structure Git Repos



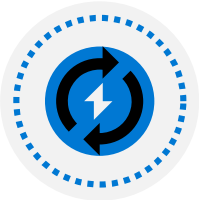
Describe Git branching workflows



Leverage pull requests for collaboration and code reviews



Leverage Git hooks for automation



Use Git to foster inner source across the organization

Module 4 review questions

1

What are the three types of branching?

2

What are Git hooks?

3

What are some best practices when working with files in Git? What do you suggest for working with large files?