Git Branching and Pull Requests

Branches in Source Control Systems

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

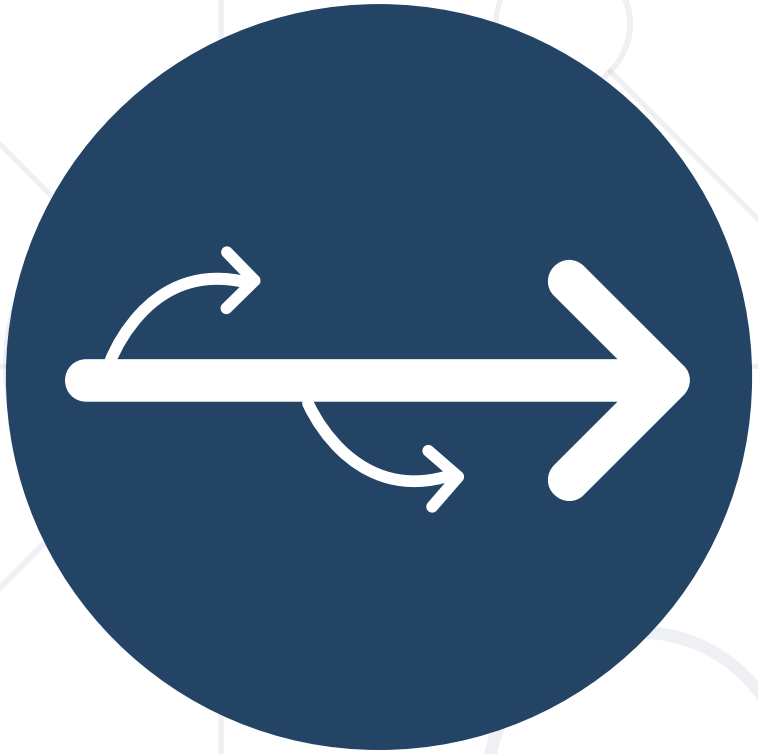
Have a Question?

Software University

sli.do

#Dev-Ops

Table of Content



Branching Concepts

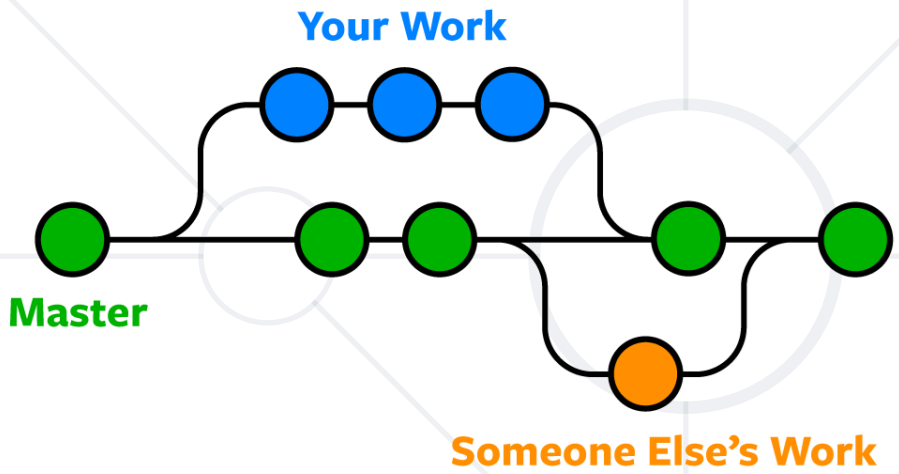
Creating and Merging Branches

What is Branching?

- **Branches** allow you to work on **different parts** of a project without **impacting** the **main branch**
  - Serve as an abstraction for the **edit / stage / commit process**
- You can **think** of **them** as a way to **request** a **brand new working directory**, staging area, and **project history**
- New **commits** are **recorded** in the **history** for the **current branch**

Branches

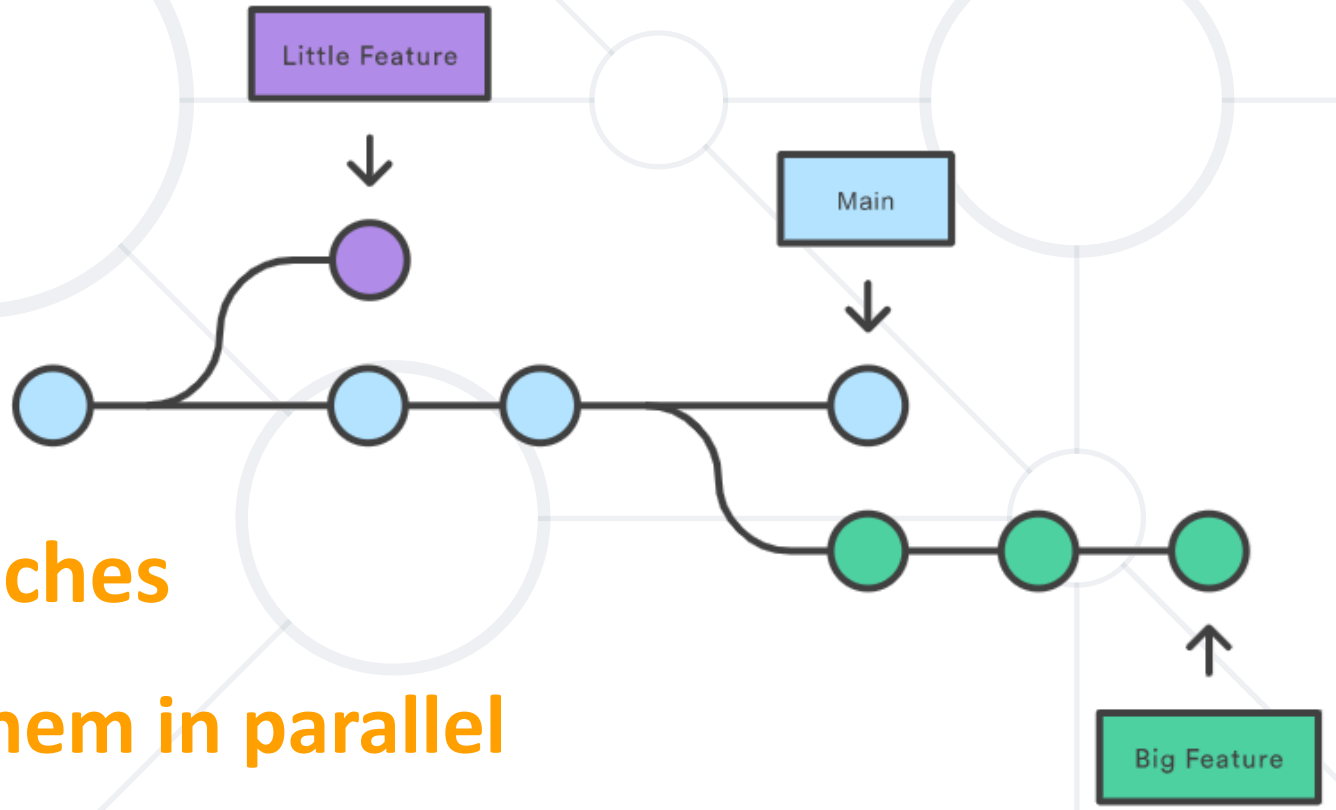
- When the **work** is **complete**, a **branch** can be **merged** with the **main project**



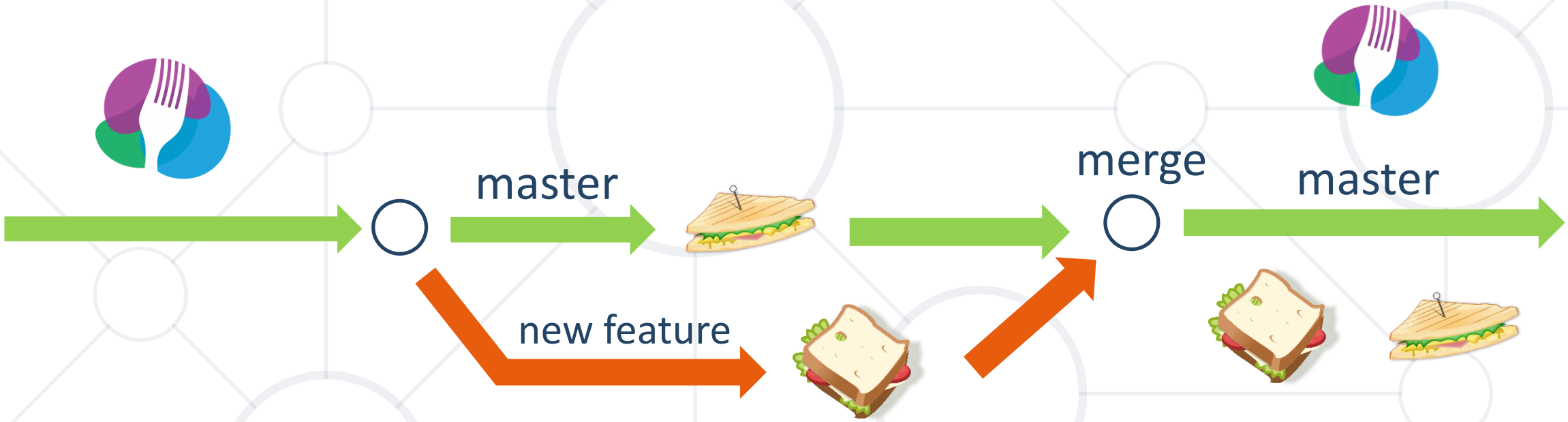
- You can even **switch between branches** and **work** on **different projects** without them **interfering** with **each other**
- **Branching** in **Git** is **very lightweight** and **fast**!

Branching – Example

- The diagram visualizes a **repository** with **two isolated lines** of development
  - One for a **little feature**
  - And one for a big, **longer-running feature**
- By developing them in **branches**
  - You can work on **both** of **them in parallel**
  - Can keep the **main branch free** from **questionable code**



Branching – One More Example



Branching in Git

Available Feature in Git

Branches are Built-In Feature of Git

- **Branching** == a **feature available** in most version control systems
- **Branching** in some version control systems can be an **expensive operation** in both time and disk space
- In **Git**, **branches** are a part of your **everyday development process**
- **Git branches** are **effectively** a pointer to a **snapshot** of your **changes**

Branching: Example

- Suppose you develop a **company website**
  - You have a **stable version**, published **live** at https://live-site.com
    - This **live site** is used everyday by hundreds of customers
  - You work on a **new site version**, which will have a **better design**
    - This takes 2-3 weeks to complete (add CSS, animations, …)
  - On the second day the team leader tells you
    - We have an **urgent bug**: contact form is broken
  - You should stop working on the new site and **fix the bug** in the live site immediately

Variant 1: Without Branches

- If you **don't use branches**, you can have separate GitHub repos
  - https://github.com/user/live-site
  - https://github.com/user/new-site
- You can checkout each repo in a different local folder
  - C:\Projects\live-site
  - C:\Projects\new-site
- When the team leader calls for the urgent bug
  - Pull the live site repo, fix the bug and deploy
  - Apply the same fix in the new site and continue working on it

Variant 1: Without Branches – Drawbacks

- Working on two separate repos has **drawbacks**
  - If you **fix a bug**, it **should be fixed twice** – in both repos
  - You **cannot merge changes** between these repos
    - You can just **copy/paste files**, but complex merging will be a pain
  - Part of your history will be lost

Variant 2: With Git Branches

- You can create two **Git branches** in the same GitHub repo
  - https://github.com/user/company-site/tree/main (live site)
  - https://github.com/user/company-site/tree/new-design (new site)
- You can switch between these branches at anytime
  - `git checkout main`
  - `git checkout new-design`
- When the team leader calls for the urgent bug
  - Switch to the live site (main) branch, pull, fix the bug and deploy
  - Switch back to the new site branch and continue working on it
  - The bug fix will be merged later, when you merge the new design

Variant 2: Without Branches – Benefits

- Benefits of working on **two parallel branches** in the same repo
  - If you **fix a bug**, the fix will be merged, when you merge the branches
  - You **can merge changes** between these branches
  - Your **history** will be preserved in the change log

Demo: Git Branches

New/Separate Version of the Main Repository

Git Branching Commands

View / Switch / Delete Branches

Local vs. Remote Branches

- **Local branch**
  - Branch in your local Git repo
  - Changes tracked only locally
- **Remote branch**
  - Branch inside the remote repository (e.g. in GitHub)
- **Upstream branch**
  - Remote branch, connected to your local branch
  - When you push changes, they are sent to the upstream branch

Git Branching Commands (1)

- **Create** a new local branch
  - `git branch {branch-name}`

```
git branch add-title
```

- **Switch** to certain existing branch
  - `git switch {branch-name}`

```
git switch add-title
```

  - `git checkout {branch-name}`

```
git checkout add-title
```

Git Branching Commands (2)

- **Create** a new branch and **switch** to it
  - `git checkout -b {branch-name}`

```
git checkout -b change-colors
```

- **List local branches**
  - `git branch`

```
git branch
```

Git Branching Commands (3)

- **List all** local and remote branches

```
git branch --all
```

```
git branch -a
```

- **List** local together with the last commit message

```
git branch --verbose
```

```
git branch -vv
```

- **List all local** and **remote branches** with the last commit message

```
git branch --all --verbose
```

```
git branch –a -vv
```

Git Branching Commands (4)

- **Push to a new upstream** (in a new remote branch)

```
git push --set-upstream origin {branch-name}
```

```
git push -u origin {branch-name}
```

- Merge another branch in the active branch
  - **git merge {branch-name}**

```
git merge add-title
```

Git Branching Commands (5)

- Delete a local branch
  - `git branch -d {branch-name}`

```
git branch -d add-title
```

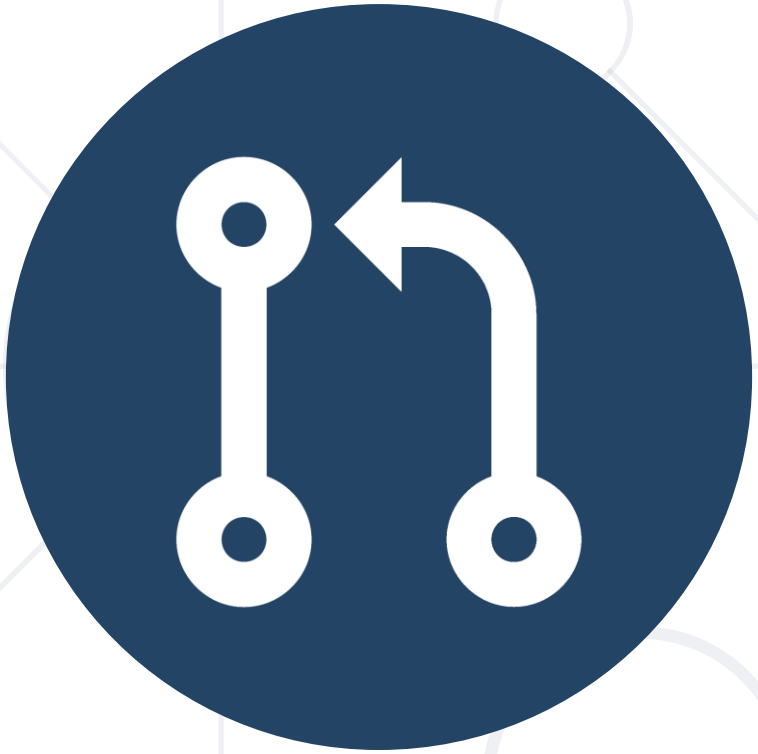
- Delete a remote branch
  - `git push origin -d {branch-name}`

```
git push origin -d add-title
```

List All Branches (Verbose)

Software University

```
C:\Work\site-demo>git branch --all -vv
  add-contacts                 a3b6a04 [origin/add-contacts] added title + css
  add-title                    55fc61e [origin/add-title] change
  change-colors                0d60c90 [origin/change-colors] changed README
  cleanup                      67ab8c4 [origin/cleanup] cleanup branch created
  main                         a3b6a04 [origin/main] added title + css
* nakov1604                    a3b6a04 added title + css
  prod                         a3b6a04 [origin/prod] added title + css
  test1537                     a3b6a04 [origin/test1537] added title + css
  test1541                     a3b6a04 [origin/test1541] added title + css
  remotes/origin/HEAD          -> origin/main
  remotes/origin/add-contacts  a3b6a04 added title + css
  remotes/origin/add-title     55fc61e change
  remotes/origin/change-colors 0d60c90 changed README
  remotes/origin/cleanup       67ab8c4 cleanup branch created
  remotes/origin/main          a3b6a04 added title + css
  remotes/origin/prod          a3b6a04 added title + css
  remotes/origin/test1537      a3b6a04 added title + css
  remotes/origin/test1539      a3b6a04 added title + css
  remotes/origin/test1541      a3b6a04 added title + css
```



Branching Strategies

Strategy Simple Branch

What is a Branching Strategy?

- **Branching strategy** == a strategy that programmers adopt while **writing**, **merging** and **deploying code** when using a **VCS**
- It is essentially a **set** of **rules** that **developers** can **follow** to stimulate how they **interact** with a **shared codebase**
- It enables teams to **work** in **parallel** to achieve **faster releases** and **fewer conflicts**
  - By creating a **clear process** when making changes to **source control**

Purposes of Branching Strategy

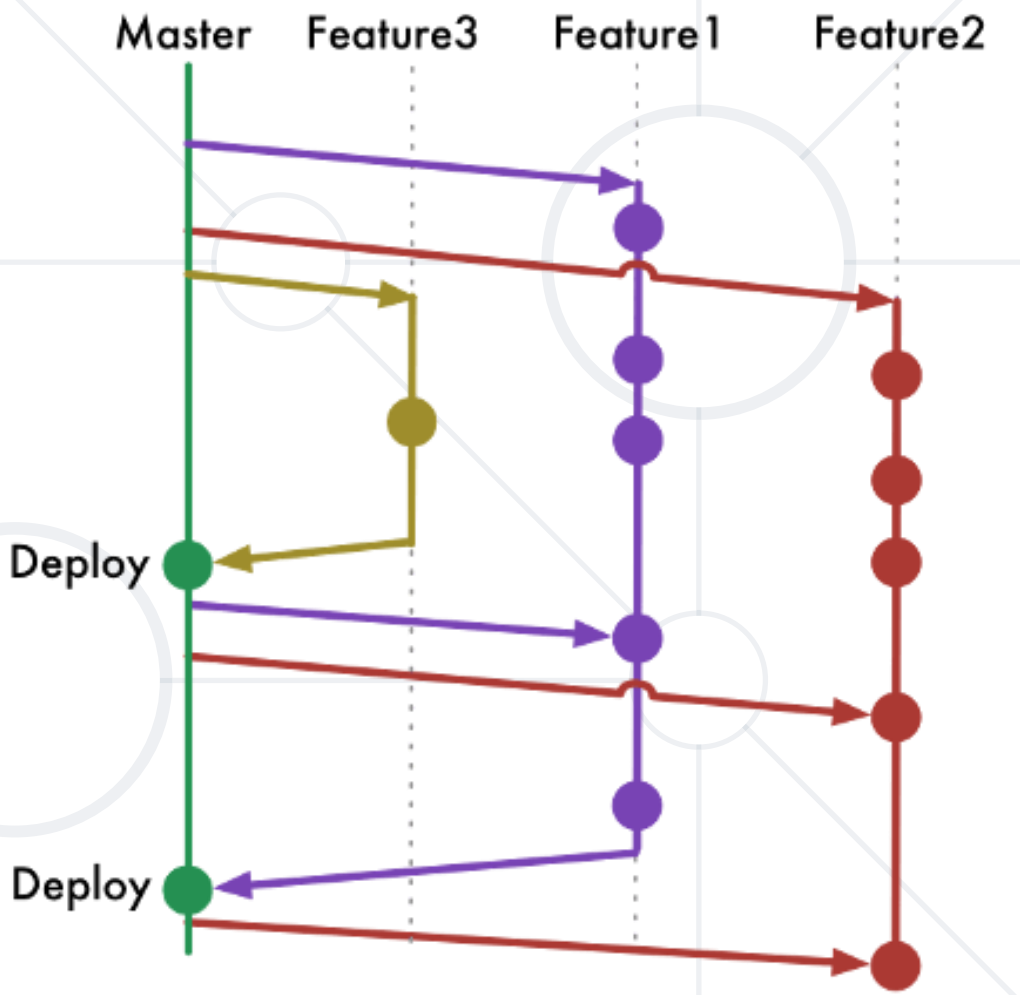
Software University

- Enhance **productivity** by ensuring proper coordination among developers
- Help **organize** a series of **planned**, structured releases
- Map a **clear path** when making changes to **software** through to **production**
- Maintain a **bug-free code**
- Enable **parallel development**

Common Git Branching Strategies

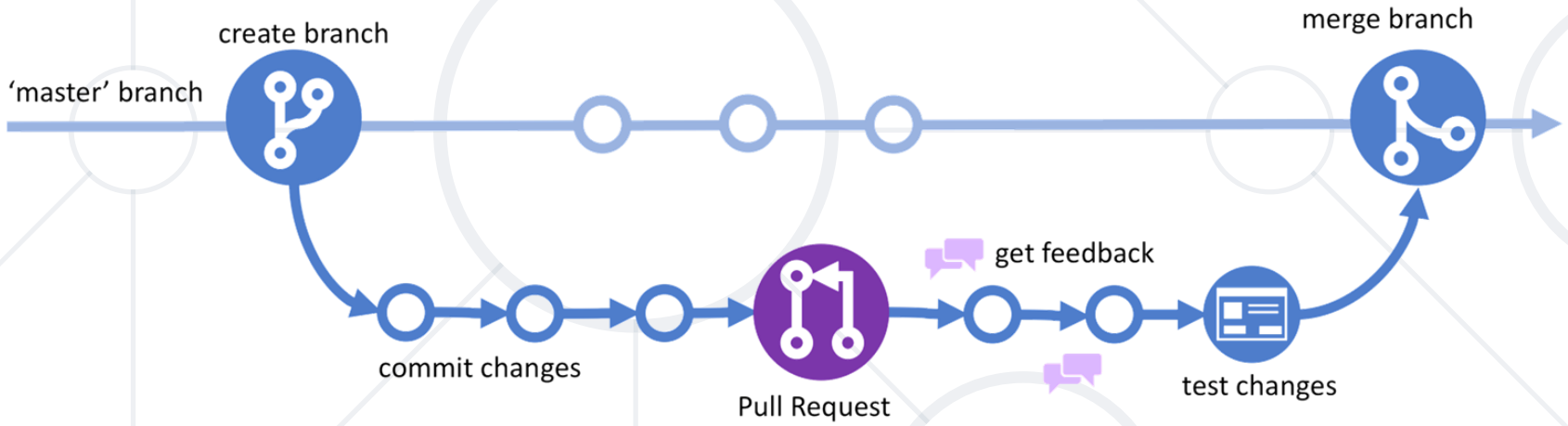
Trunk-Based Development

- **Trunk-based development**
  - "**No branches**" strategy
  - **Developers** integrate their **changes** into a **shared trunk** at **least once** a **day**
- Make smaller **changes** more **frequently and** commit **directly** into the **trunk** (without the **use** of branches)
  - This shared **trunk** should be ready for **release anytime**
- This **strategy** is **suited** to more **senior developers**



GitHub Flow

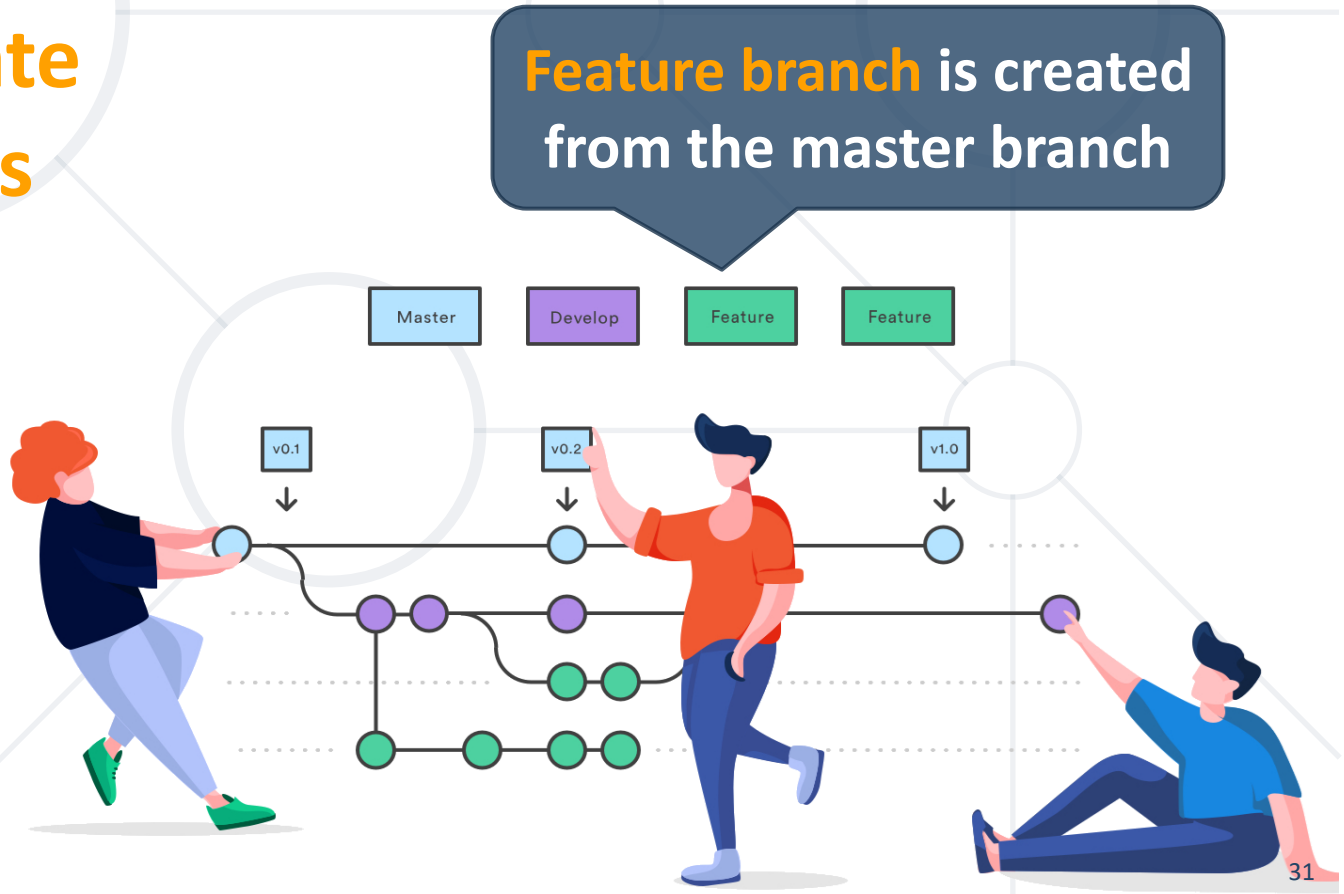
- **GitHub Flow** == **lightweight** and **flexible** development **workflow**



  - **Each feature** is implemented in its **own branch**
  - **Keep** the **master code** in a **constant deployable state**
- **Before merging**, each branch comes through a **Pull Request** and **code review**, then the **changes are tested** and integrated

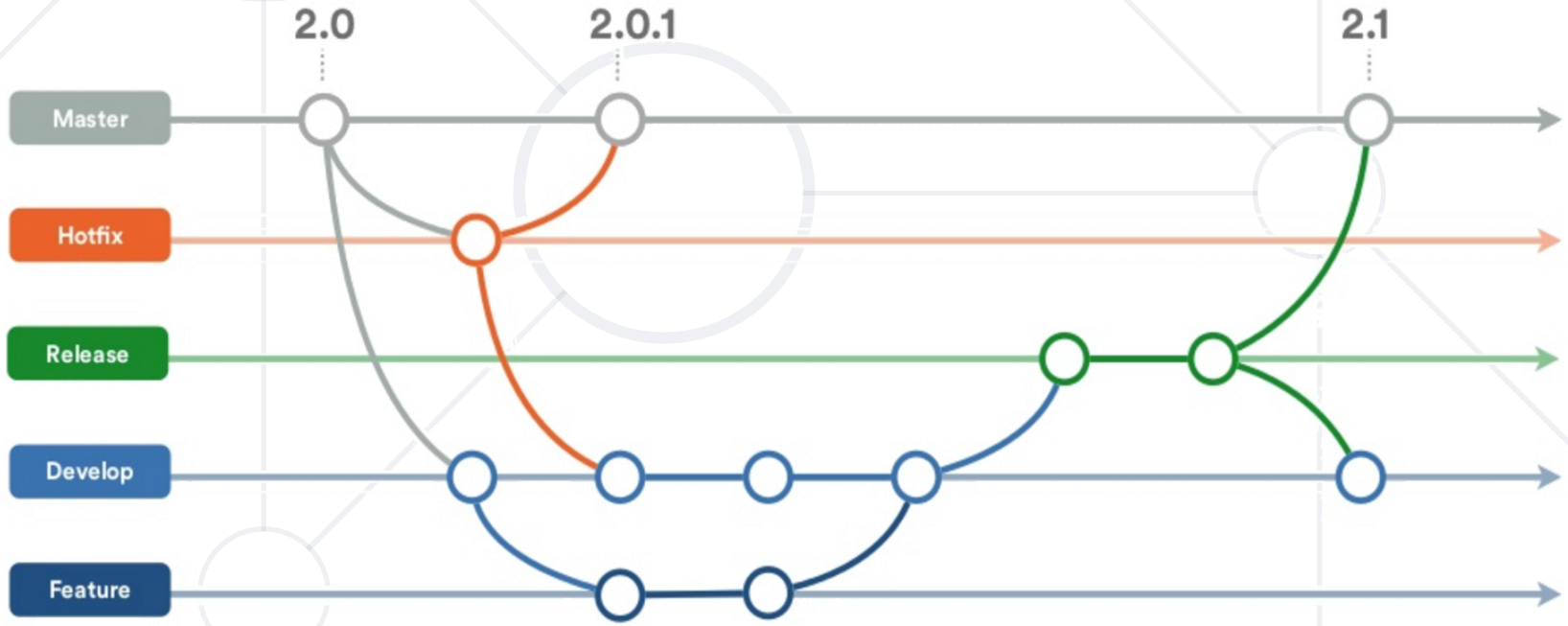
GitFlow

- **GitFlow** enables **parallel development**
  - Devs can work **separately** from the **master branch** on **features**
- This **strategy contains separate** and **straightforward branches** for **specific purposes**
  - Ideal to **handle multiple versions** of the **product**



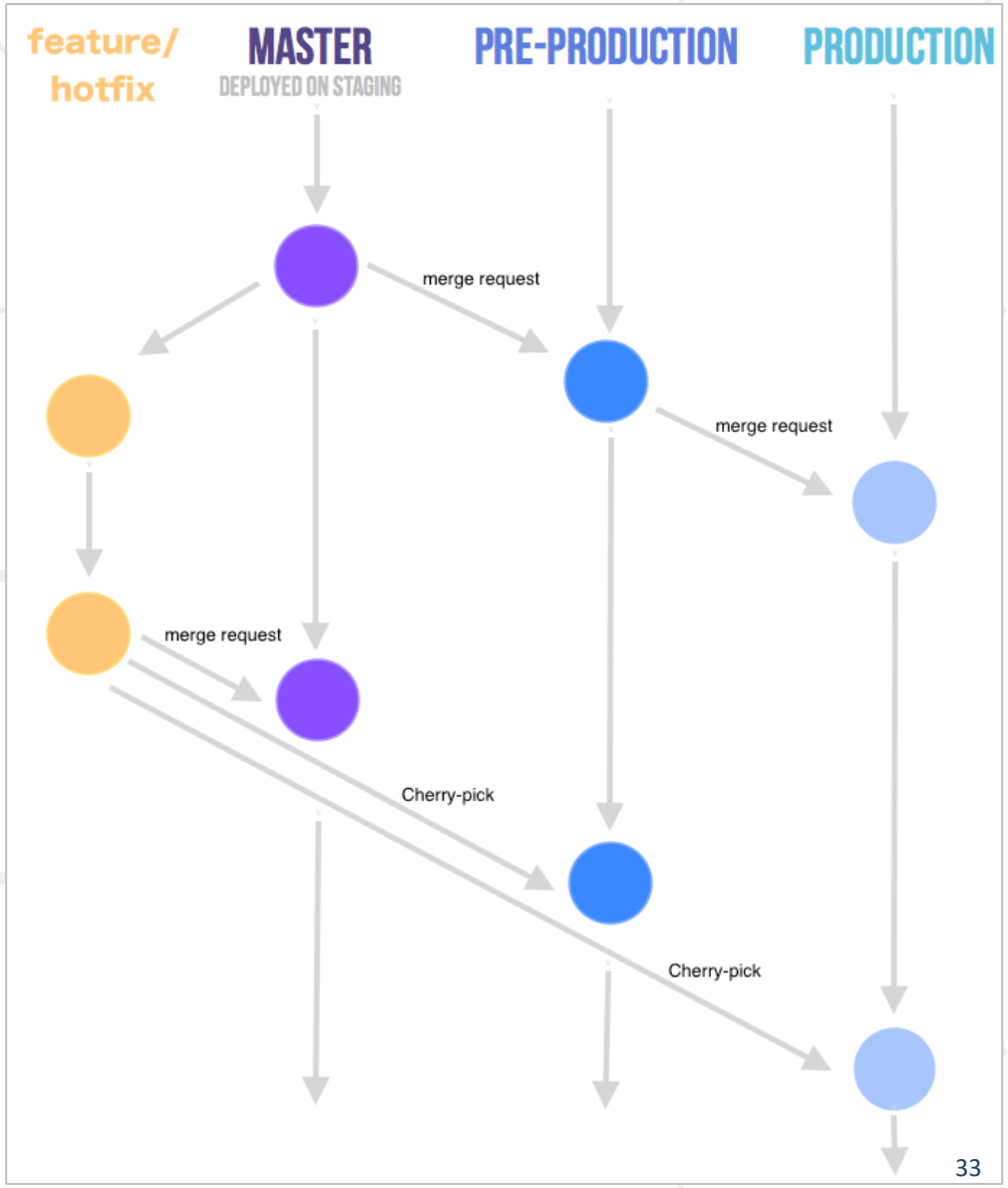
GitFlow Branches

- **Master** (**main**) – holds the active production code
- **Release** – **help** prepare a **new production release**
- **Hotfix** – hotfix branches arise from a **bug** that has been **discovered** and must be **resolved**
- **Develop**
- **Feature** – develop **new features**



GitLab Flow

- **GitLab Flow** combines **feature-driven development** and **feature branching** with **issue tracking**
  - Each **feature** is implemented in its **own branch**
  - Developers work with the **main branch right away**
  - The **main** branch is deployed to **production** (manually or auto)



Best Branching Strategy for Your Team

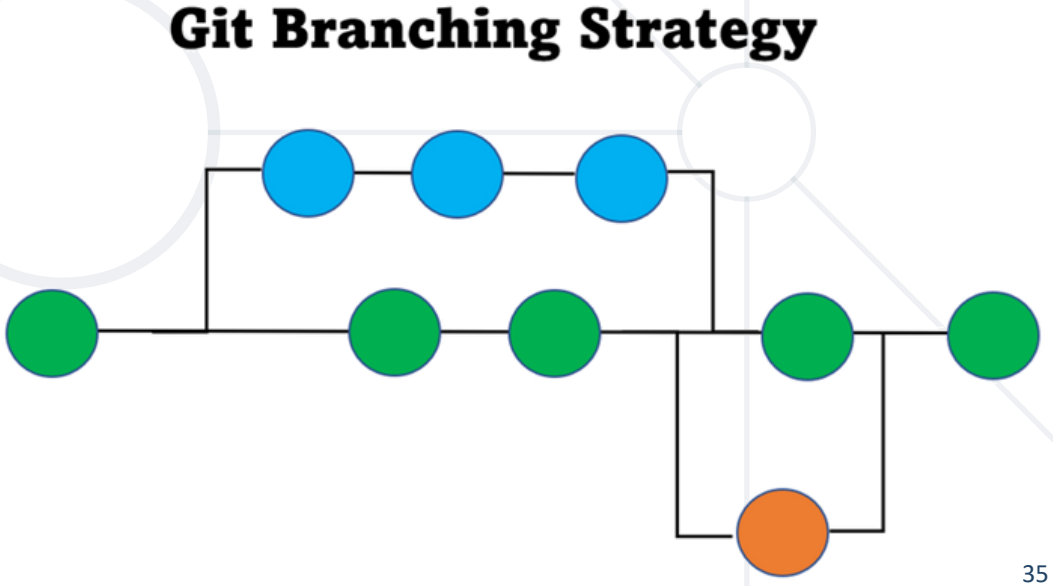
Software University

Product type and its release method	Team size	Collaboration maturity	Applicable mainstream branch mode
All	**Small team**	High	**Trunk-based Dev**
Products that support continuous deployment and release	**Middle**	Moderate	**GitHub-Flow** and **Trunk-based Dev**
Products with a definite release window and a periodic version release cadence	**Middle**	Moderate	**Git-Flow** and **GitLab-Flow** with **release branch**
Basic platform products	**Middle**	Moderate	**GitLab-Flow**
Products demanding product quality and have a long maintenance cycle for released versions	**Large**	Moderate	**Git-Flow**

No Standard for "Branching Strategy"

Software University

- Different organizations use **different branching strategies** for each different team / project
  - There is no "***best branching strategy***"
  - How to use branches highly depends on the **team** and on the **project** under development



Pull Requests in GitHub

Show Changes You've Pushed

Pull Requests in GitHub

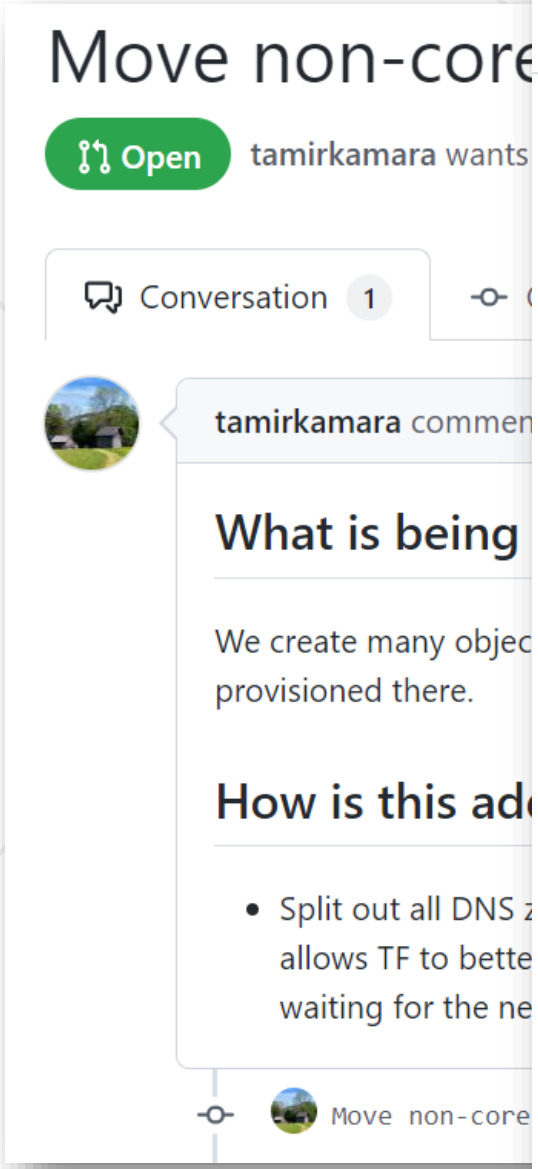
- **Pull requests** == a **mechanism** for a developer to **notify** their team members that they have **completed** a **feature**
- The **pull request** is more than just a **notification**—it's a **code review process**, including **discussions** the **proposed feature**
- If there are any **problems** with the **changes**, **teammates** can **post feedback** in the **pull request**
- This **activity** is **tracked** directly **inside** of the **pull request**

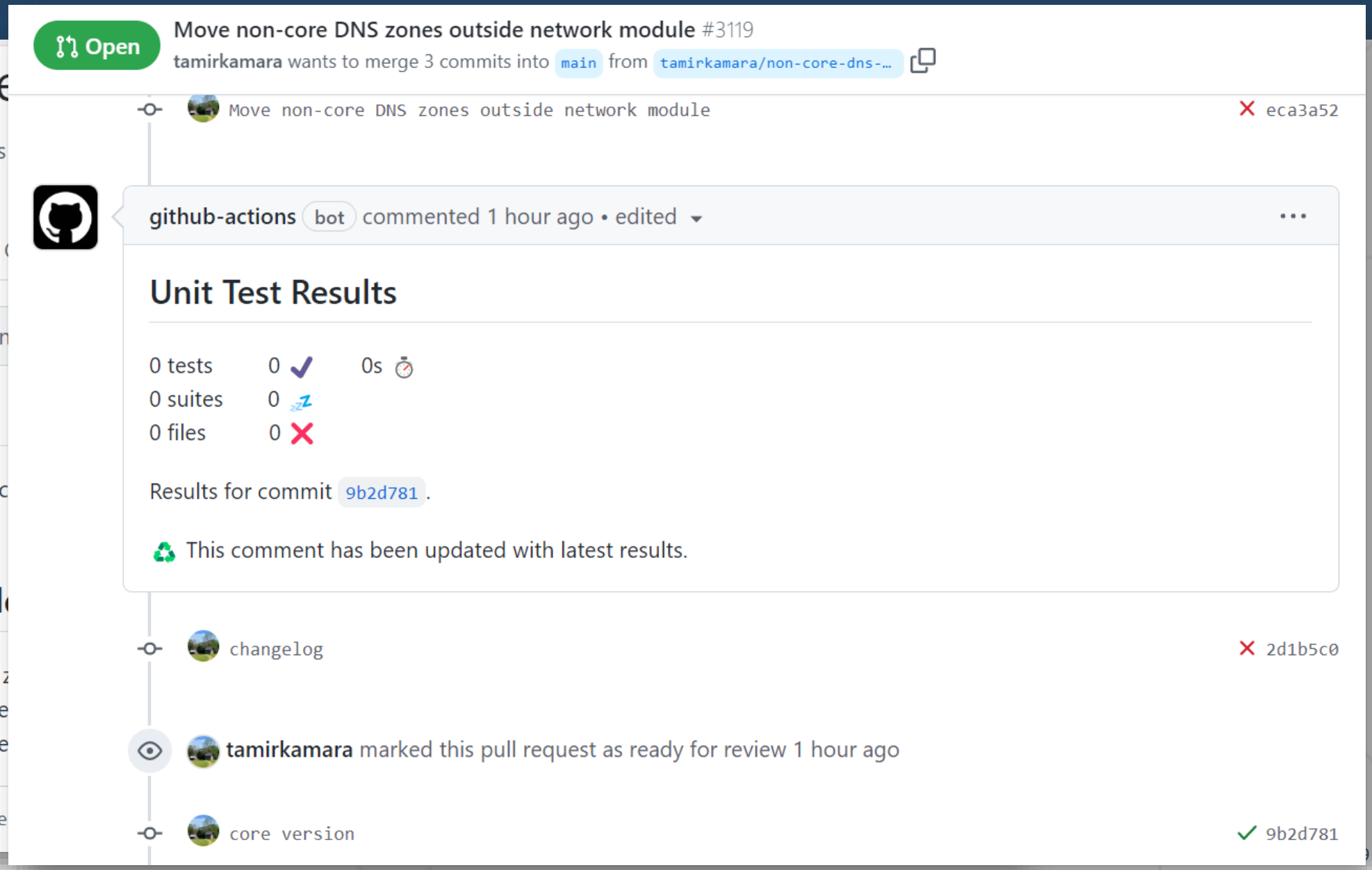
Pull Requests

- **Pull requests** let you tell others about **changes** you want to **push** to a **branch** in a **repository** on GitHub
- Once a **pull request** is **opened**, you can **discuss** and **review** the **potential changes** with **collaborators**
- You can **create pull requests** on **GitHub.com**, in **GitHub Desktop** or in other GitHub tools

Pull Request – Examples

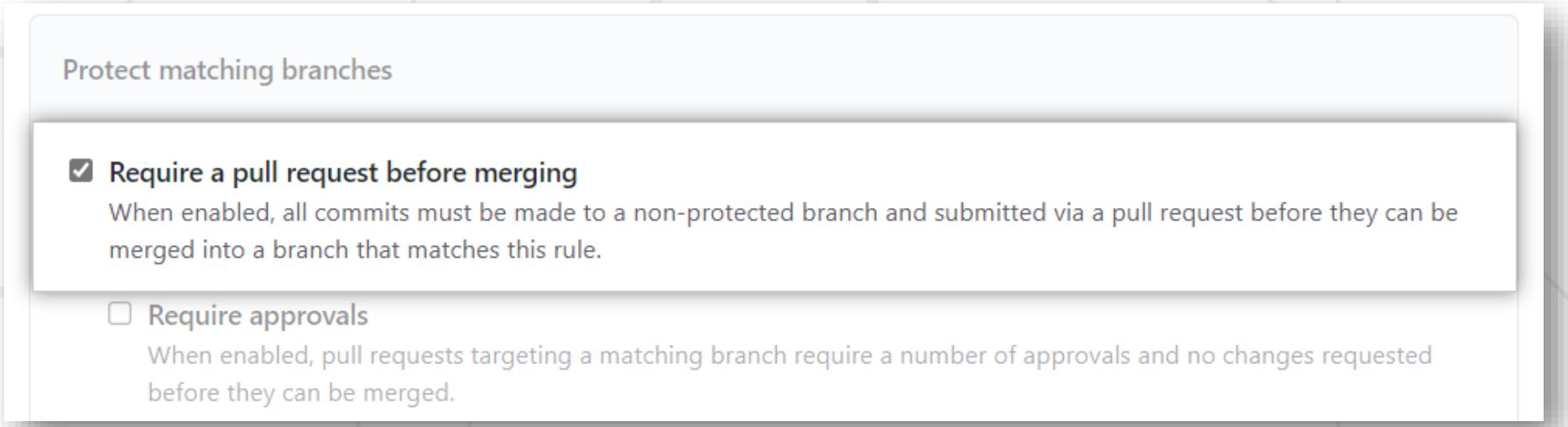
Software University





Branch Protection Rules

- You can create a **branch protection rule** in a repo
  - Can be created for a **specific branch**, all **branches**, or any branch that **matches** a **name pattern** you specify with **`fnmatch` syntax**
- For example, you can create a branch protection rule to prevent **contributors from merging to a branch** without an **approved pull request**



The Pull Request Process

Notify Team for a Completed Feature

Pull Request: The Process

Programmer

Senior

- Create a feature branch
- Make and commit changes
- Open a pull request
- Resolve merge conflicts
- Request a review

Discussion

- Give feedback / comment

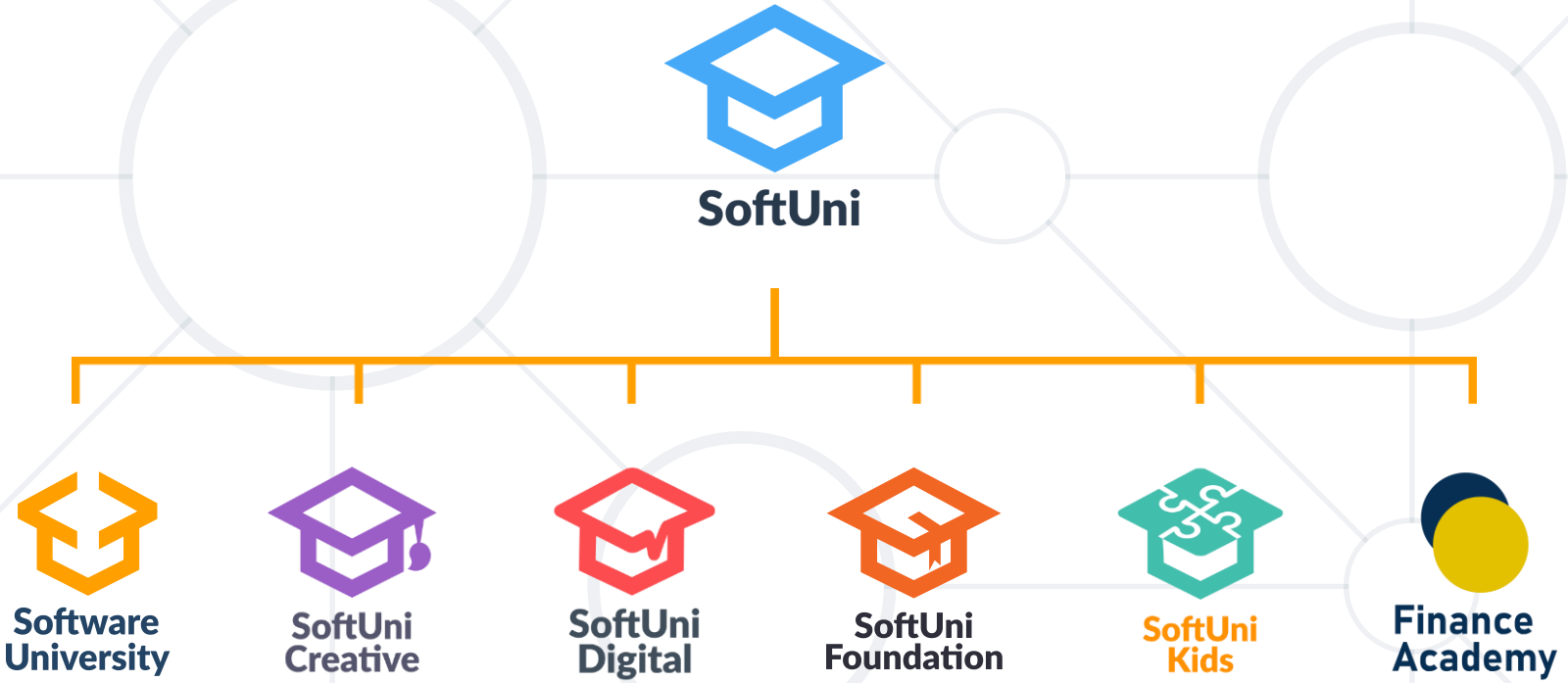
- Approve and merge

Demo: Creating a Pull Request

Summary

- **Branch** == a **new separate** version of the **main repository**
- Build your branching strategy from these **three main concepts**
  - Use **branches** for all **new features** and **bug fixes**
  - **Merge feature branches** into the main **branch** using **pull requests**
  - Keep a high quality, up-to-date **main branch**
- **Pull requests** let you tell others about changes you've pushed to a **branch** in a **repository** on **GitHub**

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

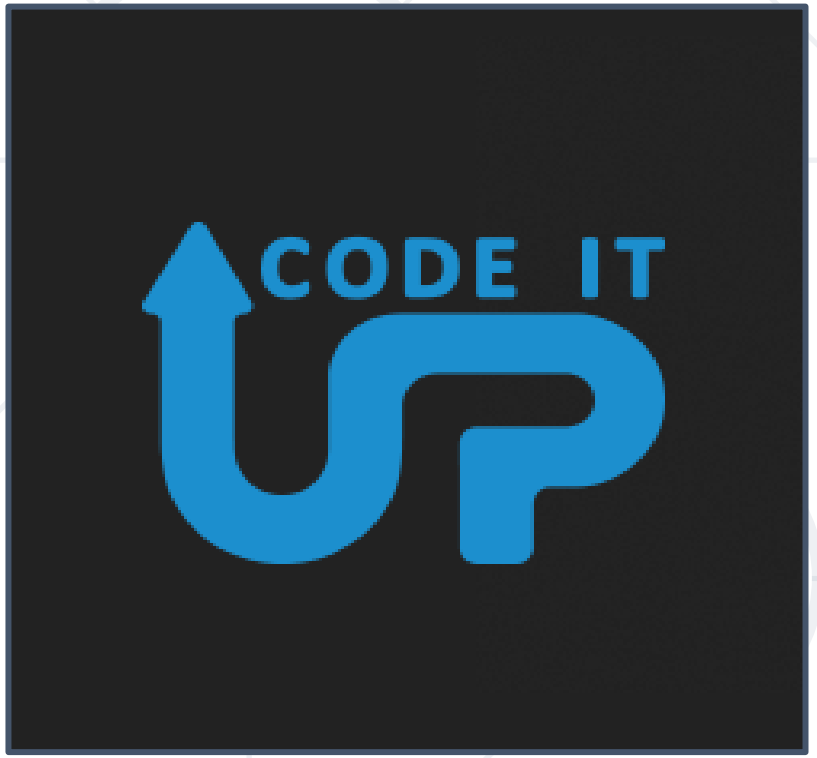
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg