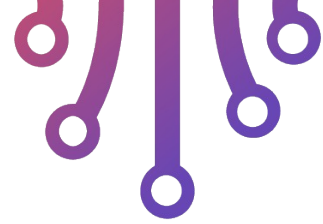


DevOps

SC-Camp 2021

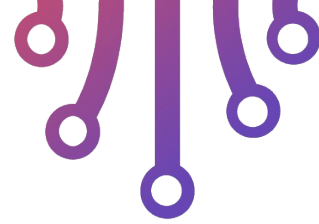


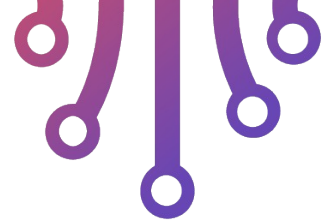
Pedro Velho

PhD - UGA 2004-2010

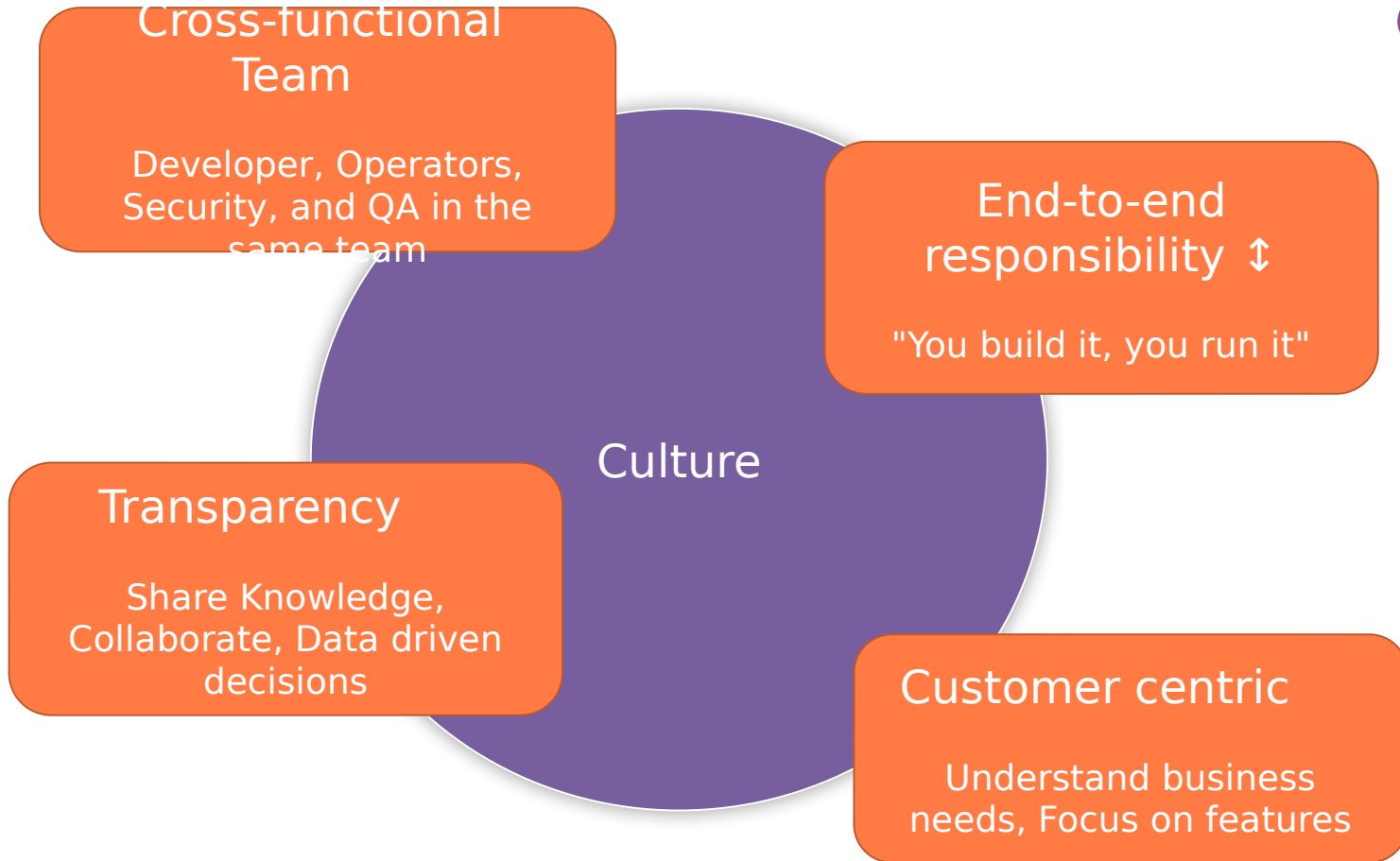
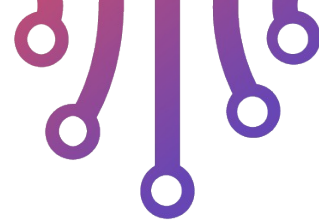
...

Engineer - Ryax - since 2019

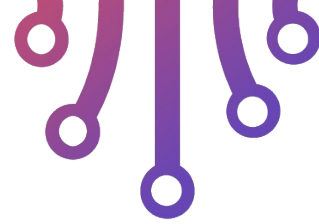




Introduction to DevOps



Organization Transformation



Functional team:

- Focus on technologies
- Long V cycle of release
- Lots of communication friction
- No understanding of user needs
- No knowledge sharing or ownership

Developer Team

Security/QA Team

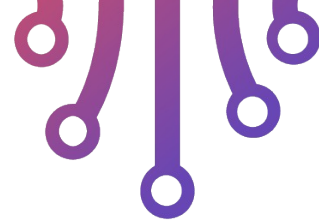
Operation Team

Product Team:

- Focus on product
- Fast Agile cycle of release
- More efficient communication
- Real understanding of user needs
- Share knowledge and ownership

Product A Team

Product B Team



Agile Methodologies

Have rituals to synchronize a collaborate efficiently

Automate ⚙️

Automate every steps with CI/CD

Infrastructure as Code

Build reproducible and immutable containers and infrastructure

Shift left

Find bug and security issues early with automated tools

Principles And Good Practices

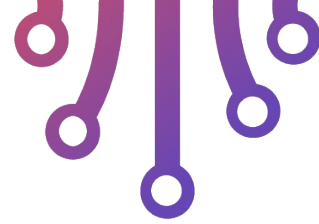
⚡ Fast Feedback

Observability: logs, metrics, and traces

Release often, release early

Quick feature release and bug fixes

DevOps Best Practices



Agile methodologies

1. Define your rituals (Daily, review, postmortem, retrospective, ...)
2. Randomly choose a Scrum Master
3. Design feature with the whole team, PO brings only business needs
4. Iterate and improve

Shift left

5. Create unit tests and integration
6. Select code quality tools
7. Put tests and quality checks in the CI pipeline
8. Only integrate code if all the quality is good

Automate:

9. Track all manual/implicit processes
10. Choose a documentation tool
11. Document them
12. Automate them if possible

Infrastructure as Code

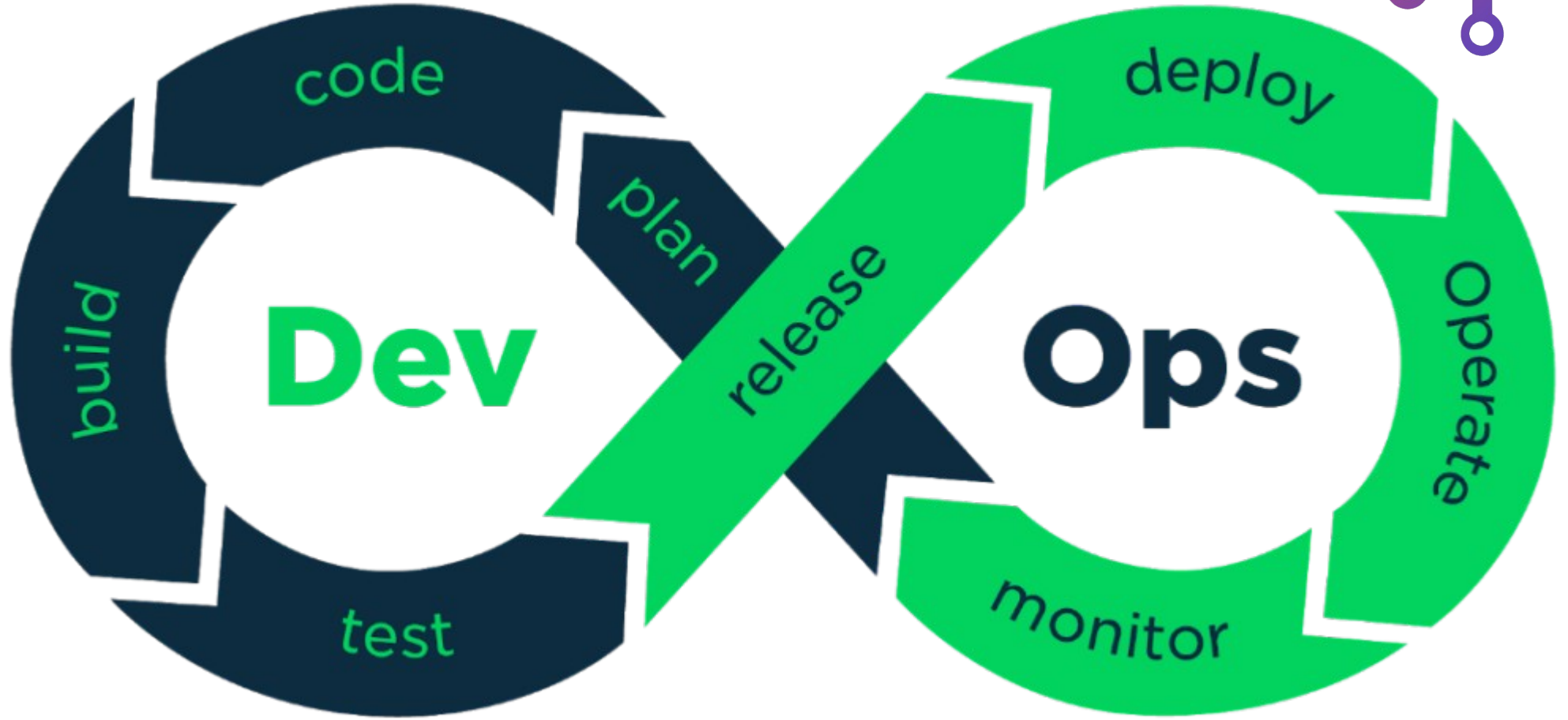
1. Choose appropriate tool
2. Code your infrastructure
3. Deploy it within the CI
4. Forbid any manual changes

Fast Feedback

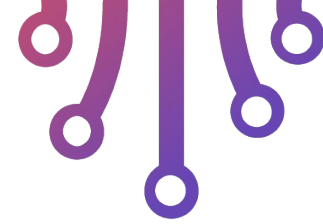
5. Get metrics, logs and traces from your apps
6. Setup alerting on user focus signals
7. Setup dashboards to observe and explore

Release often, early

8. Implement other practices
9. Automate deployment with CD
10. Monitor and rollback if necessary
11. Track the time-to-deliver and try to improve it!



DevOps Life Cycle Tools



Plan

- Use knowledge sharing tools
Example: wiki, issue tracker, whiteboard and post-it

Code

- Use an IDE with integrated quality check tools
Example: VSCode, PyCharm
- Enforce code integration process with code review
Example: Gitlab, Github

Build

- Enforce reproducible build
Example: Poetry (Python), Yarn (NodeJs)
- Package in a container
Example: Docker, Kaniko, Nix, Buildah

Test

- Create a pyramid of test with coverage
Example: Pytest (Python)

Release

- Push build artifact to a registry with a unique version tag
Example: Skopeo, DockerHub, Harbor

Deploy

- Deploy your container(s) on a cluster
Example: Kubernetes, Nomad

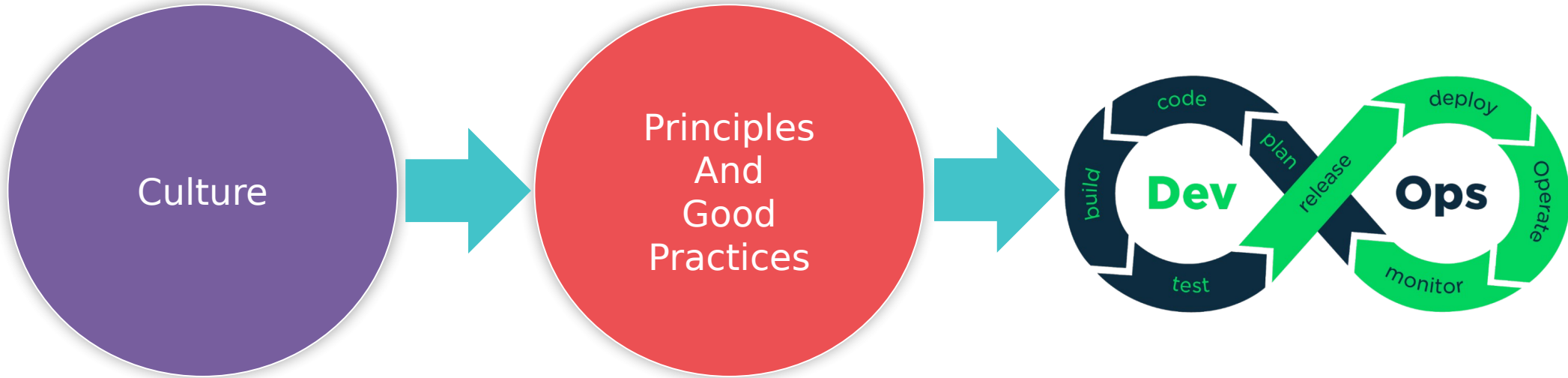
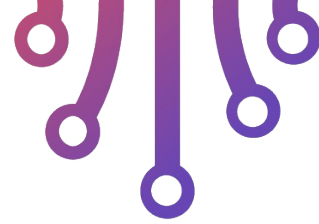
Operate

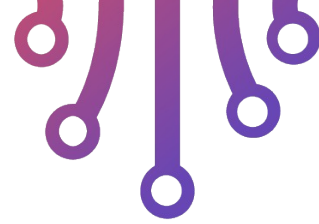
- Manage infrastructure updates
Example: Helm, Kubeadm

Monitor

- Monitor important user-facing signals (latency, traffic, errors, saturation)
Example: Prometheus, Grafana
- Setup an alerting system
Example: Prometheus, Slack

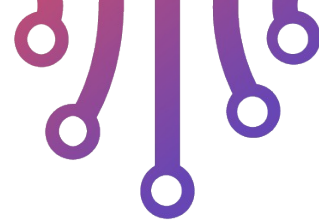
DevOps: Culture, Practices, Tooling





Versioning

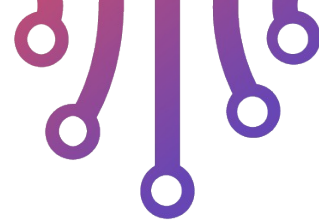
- Introduction to versioning with git
- commit
- pull/push
- merge
- why not use rebase?



Versioning

- Code Versioning Systems propose:
 - Keep track of changes, who did what and when?
- Examples:
 - Fast rewind to find code before a change/feature
 - Fast forward to a new experimental feature
 - Find first change that introduced a bug
 - Who introduced the bug and when? Blame!

Jhon Romeros talk : Doom's a year of madness : "How did you handle version control back then? There was none,... We just watched not to touch other presons' files", see full talk here <https://www.youtube.com/watch?v=eBU34NZhW7I>



Versioning

- Git, a bit of history

Other CVS exist: cvs, mercury, svn

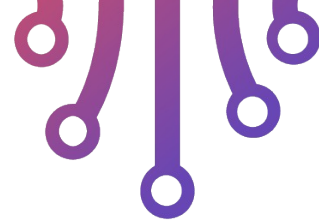
Not adapt well to the linux kernel development culture

Mainly remote devs and volunteers geographically spread

Git was one of the first to introduce CVS in a distributed manner

Heavily influenced by the non-free solution bitkeeper

Git book is free to read online <https://git-scm.com/book/en/v2>



Versioning

- Introduction to git

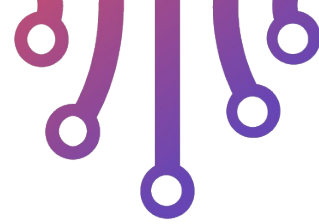
Every code change is explicit commit by the dev

The commits make a history tree (formaly a graph)

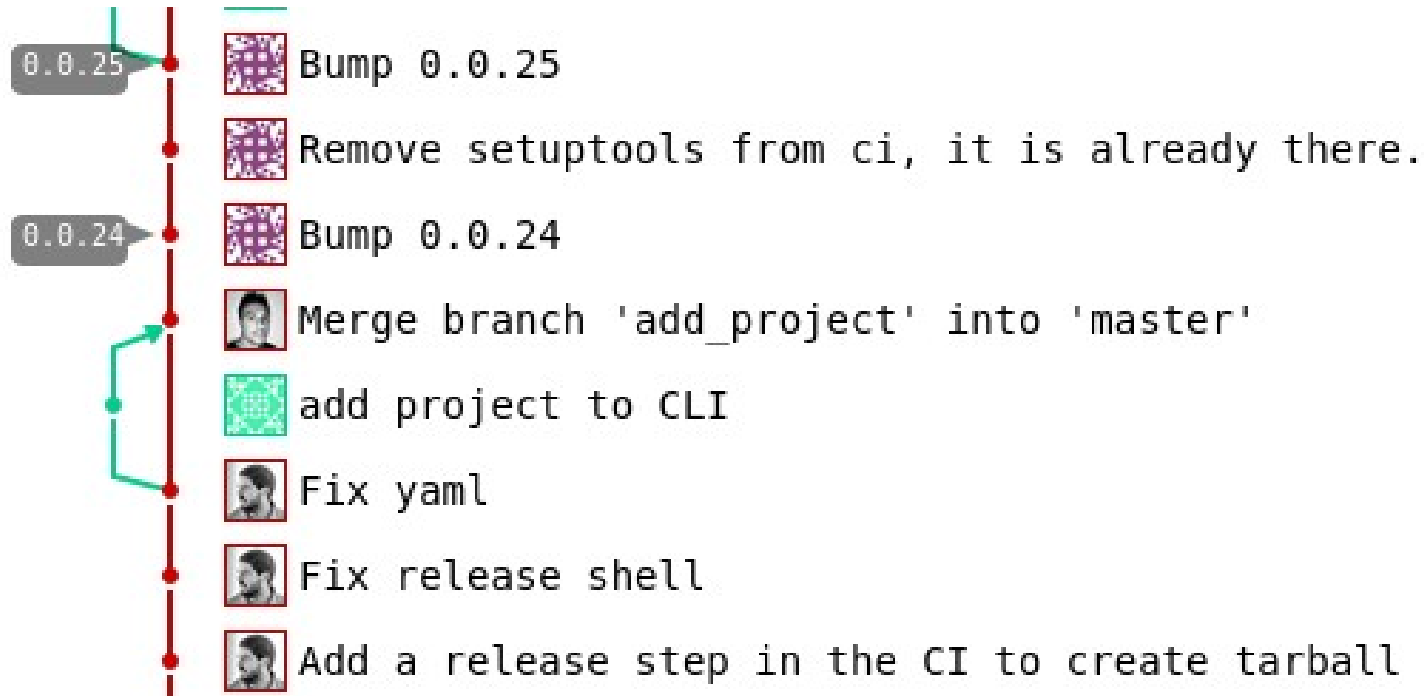
Every node on the tree is a commit (code change)

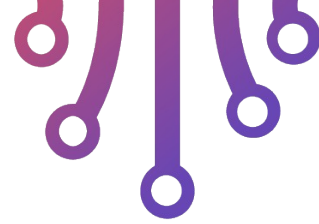
Parallel changes are allowed (branch)

Parallel changes can be merged

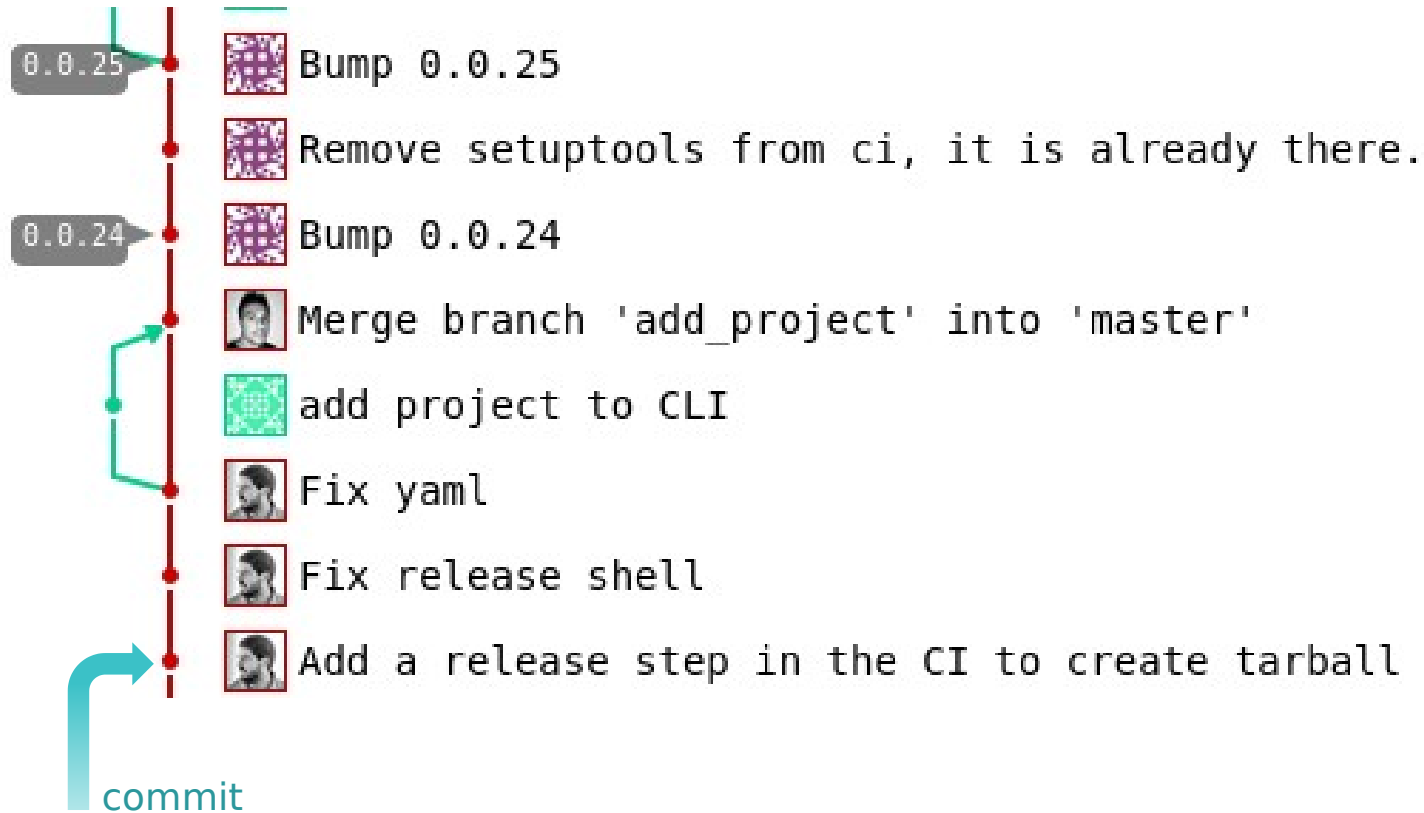


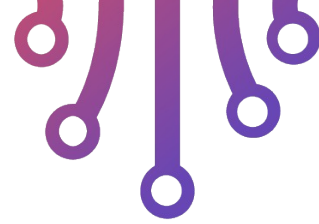
Versioning



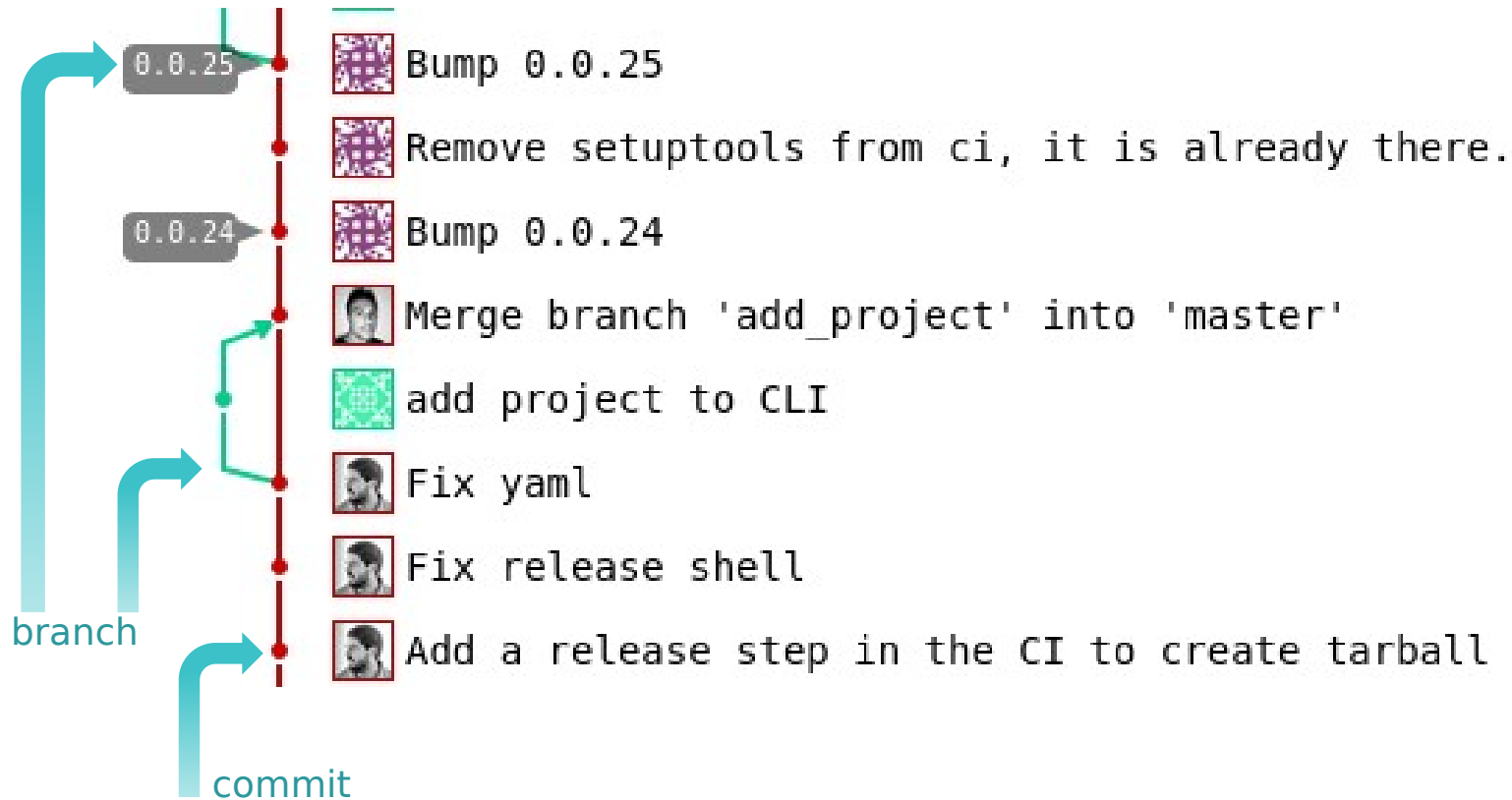


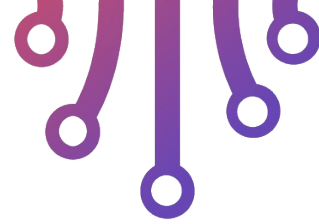
Versioning



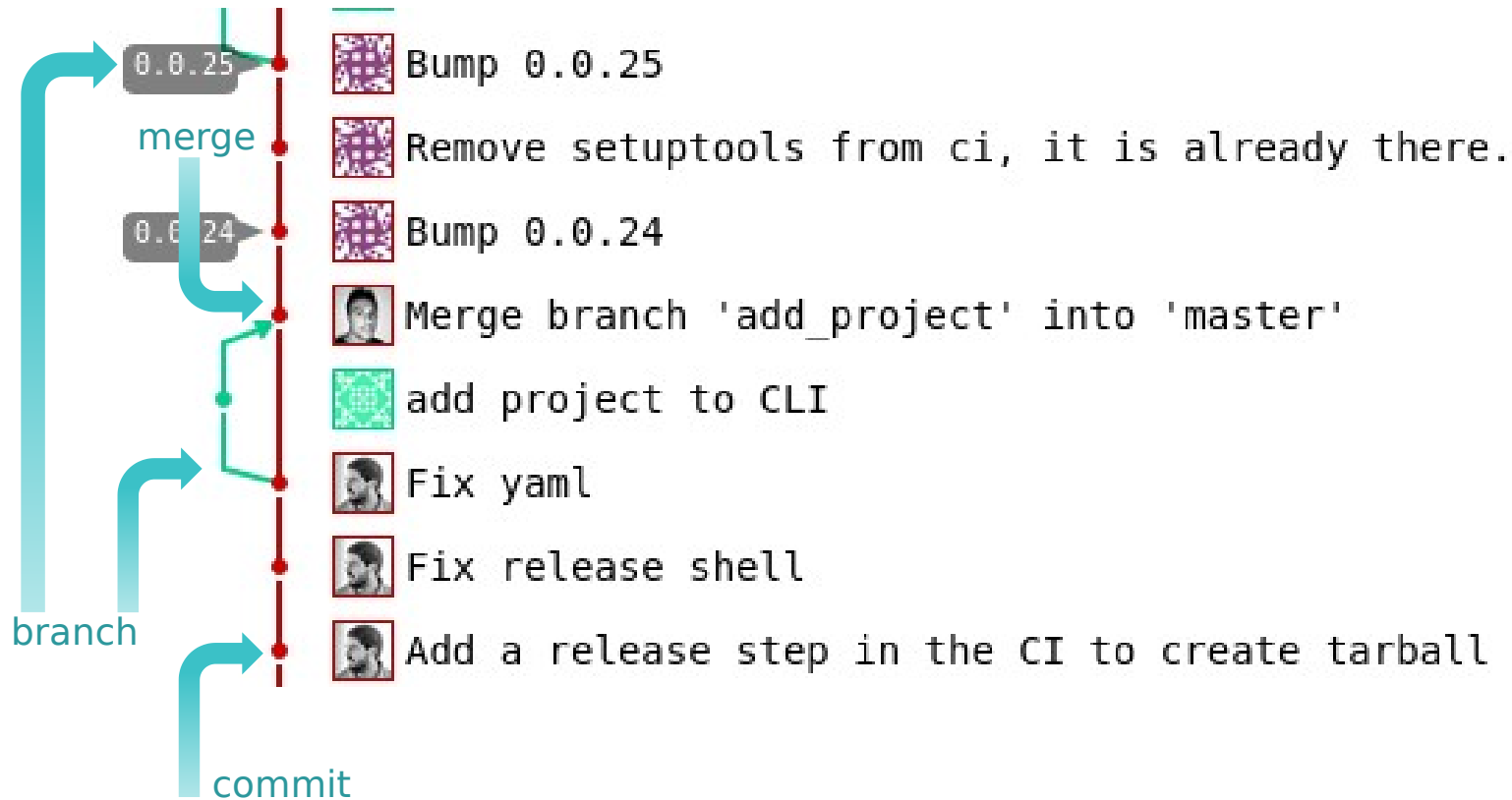


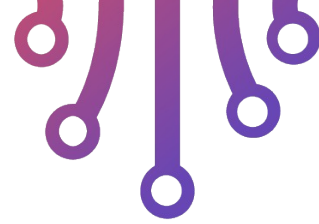
Versioning





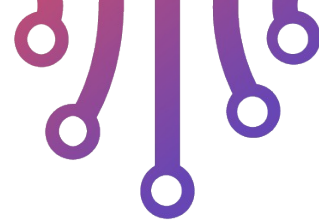
Versioning





Versioning

- The remote history (commit tree) is shared amongs devs
- Every dev copy the full history to start working
- Devs then make changes on the local history
 - Branch
 - Commit
 - Merge
- Once changes are ready devs can **push** changes to the remote
- Devs can **pull** remote changes as well
- Beware of conflicts



Versioning

- Working example 1 dev (Bob)

Clone from initial commit

```
git clone
```

Add a README.md file

```
git add README.md
```

```
git commit -m "Add README.md FILE"
```

Update README.md file

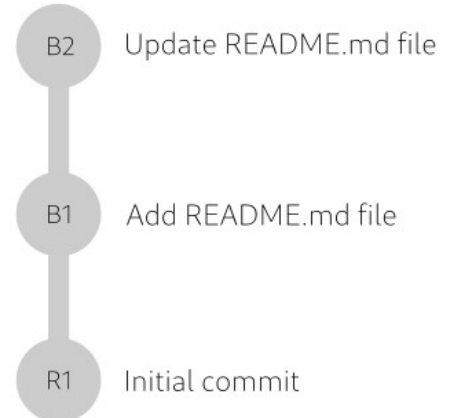
```
git add README.md
```

```
git commit -m "Update README.md file"
```

Remote history

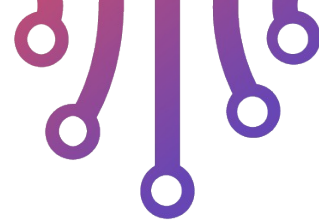


Bob's history



clone





Versioning

- Working example 1 dev (Bob)

Clone from initial commit

```
git clone
```

Add a README.md file

```
git add README.md
```

```
git commit -m "Add README.md FILE"
```

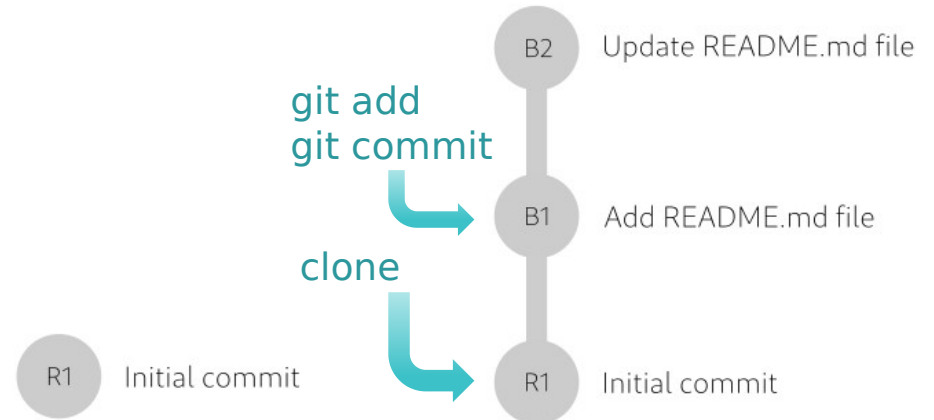
Update README.md file

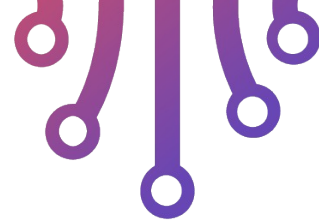
```
git add README.md
```

```
git commit -m "Update README.md file"
```

Remote history

Bob's history





Versioning

- Working example 1 dev (Bob)

Clone from initial commit

```
git clone
```

Add a README.md file

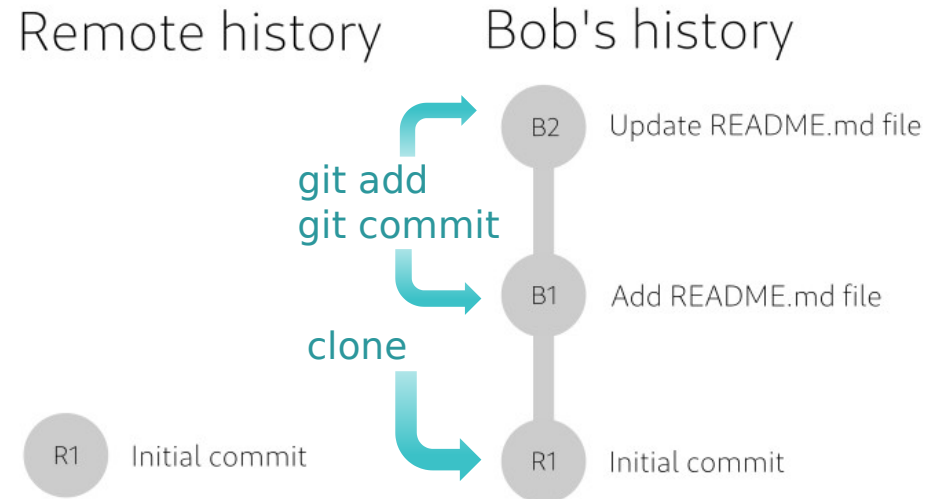
```
git add README.md
```

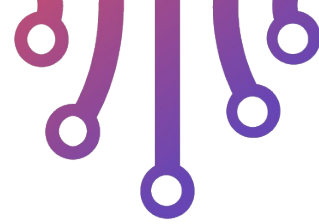
```
git commit -m "Add README.md FILE"
```

Update README.md file

```
git add README.md
```

```
git commit -m "Update README.md file"
```





Versioning

- Working example 1 dev

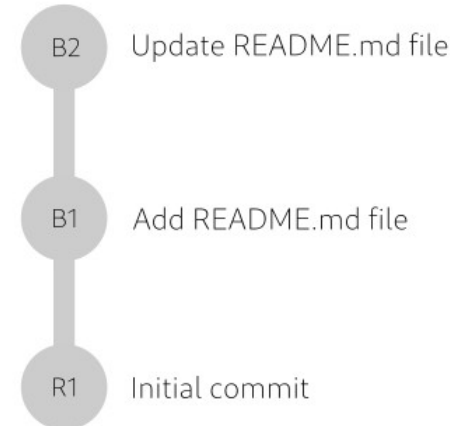
Remote history only add B1 and B2 after bob calls

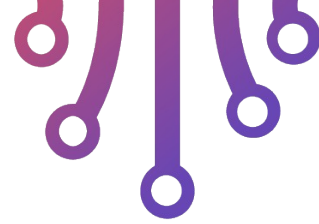
git push

Remote history



Bob's history





Versioning

- Working example 1 dev

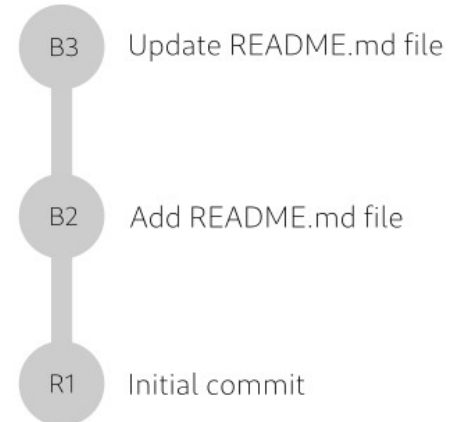
Remote history only add B1 and B2 after bob calls

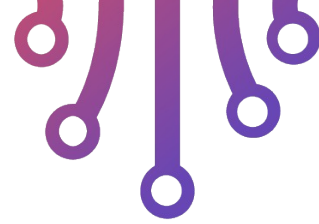
git push

Remote history



Bob's history





Versioning

- Working example 2 devs (ana & bob)

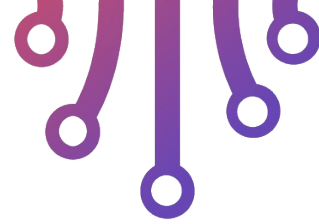
Ana start fresh clone from R1

Bob start fresh clone from R1

Bob finishes and push his work to remote upstream

Ana finishes and push her work to the upstream

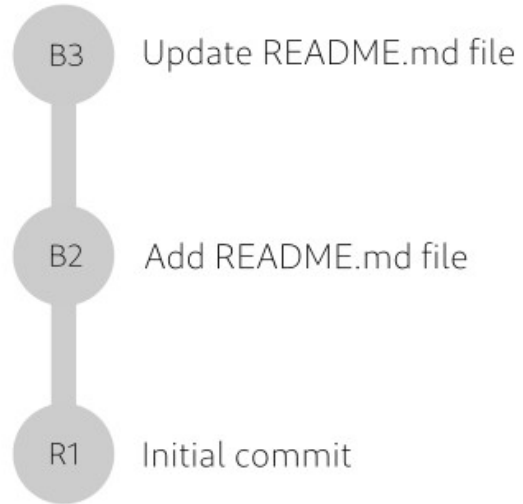
What happens?



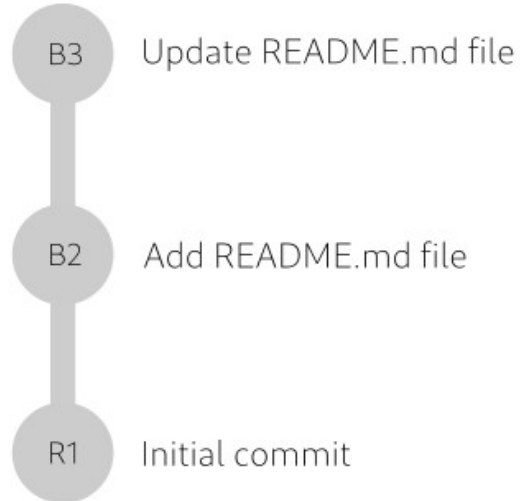
Versioning

- Working example 2 devs (ana & bob)

Remote history

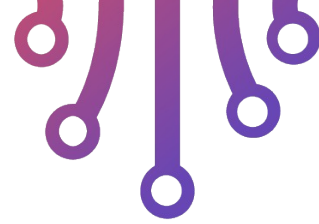


Bob's history



Ana's history

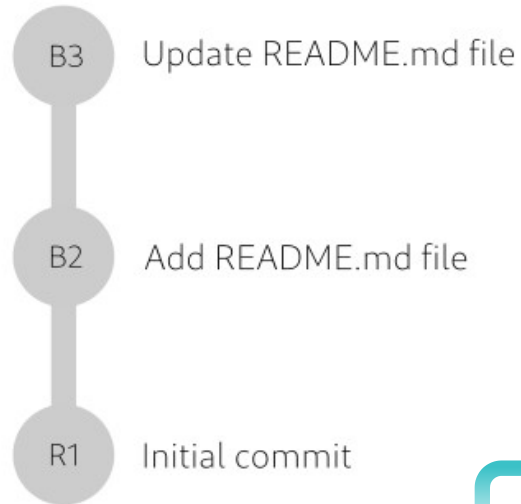




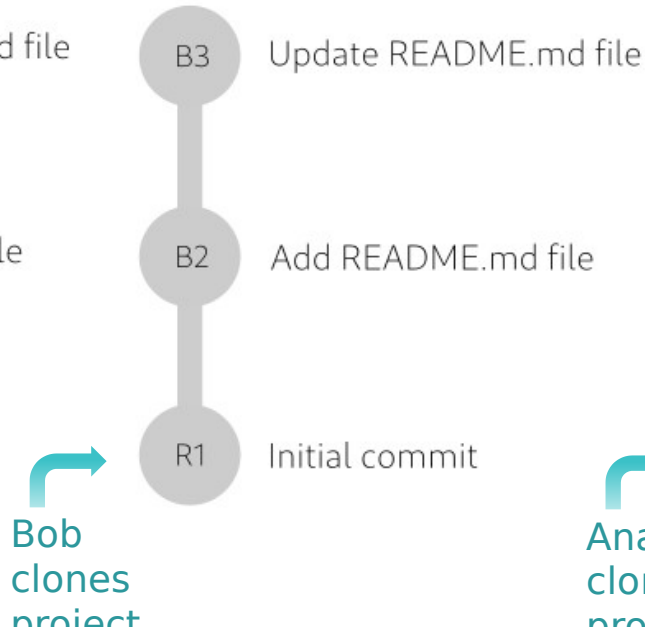
Versioning

- Working example 2 devs (ana & bob)

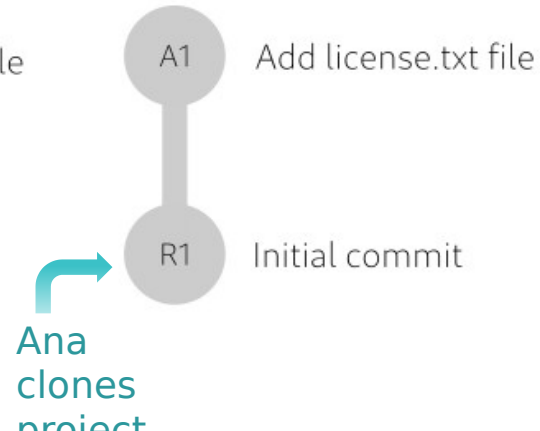
Remote history

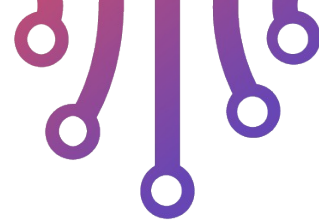


Bob's history



Ana's history

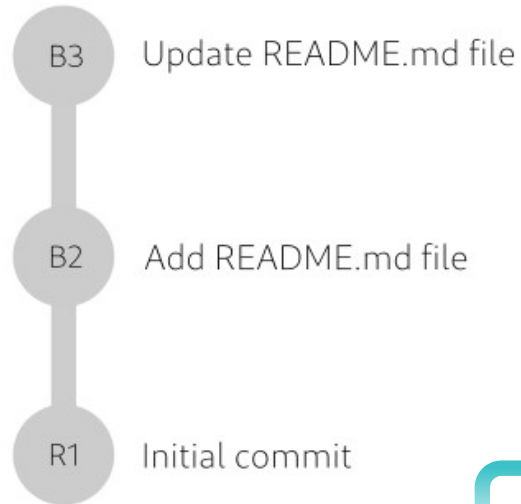




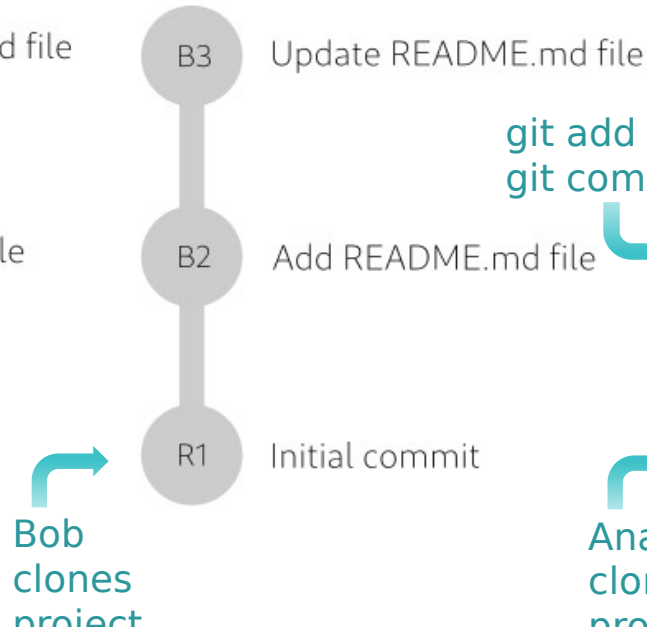
Versioning

- Working example 2 devs (ana & bob)

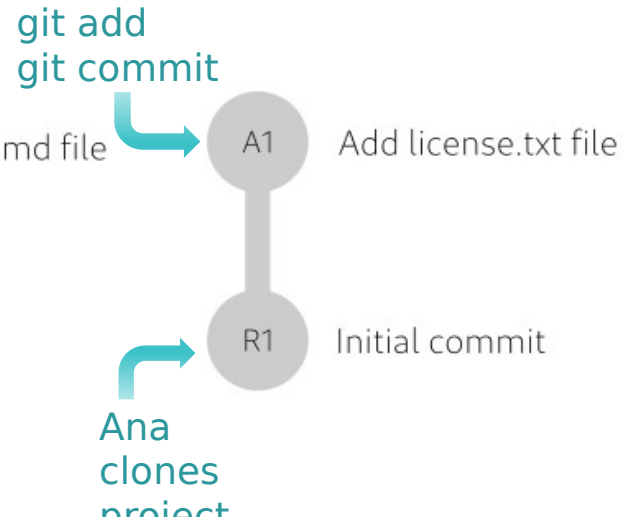
Remote history

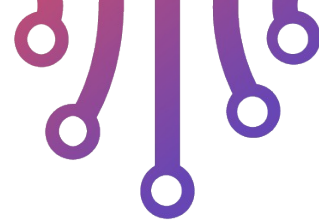


Bob's history



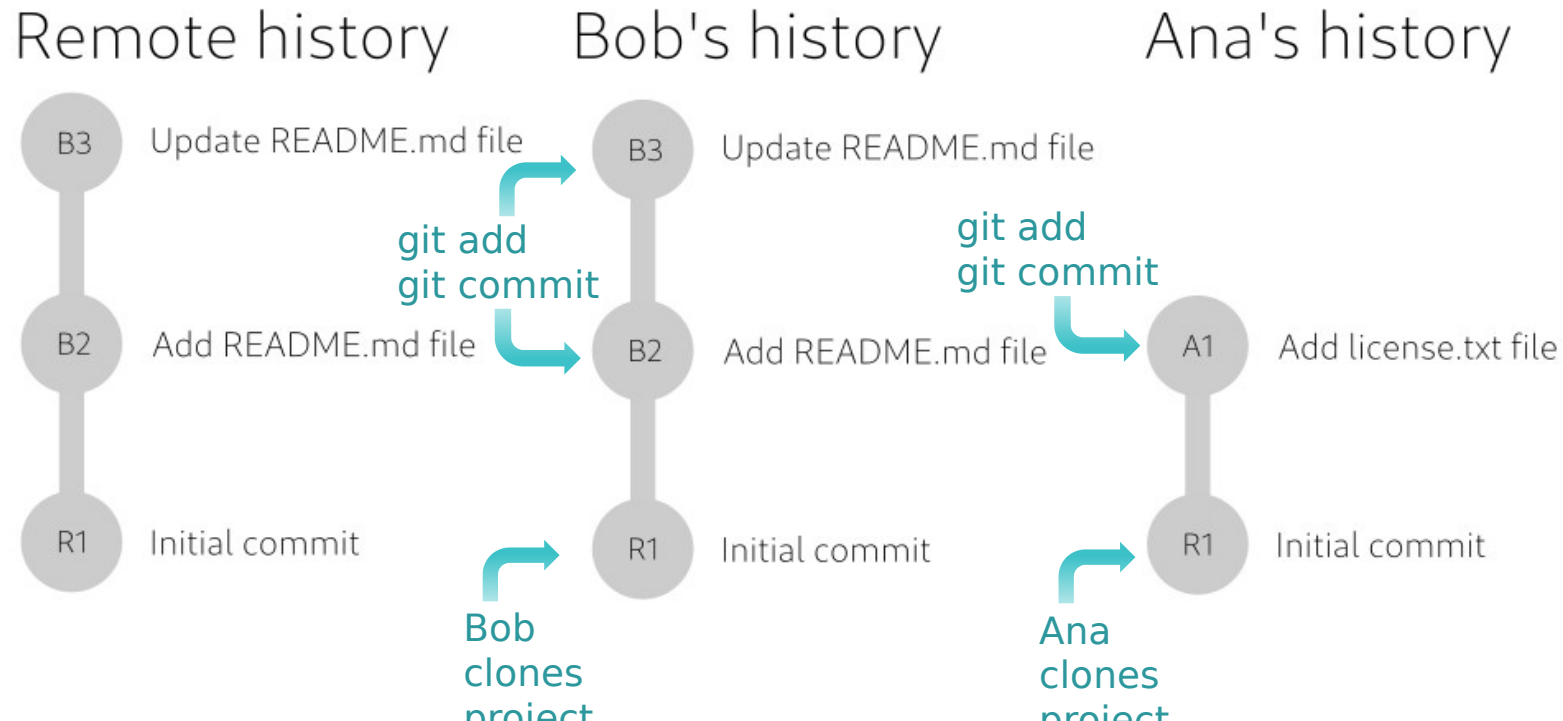
Ana's history

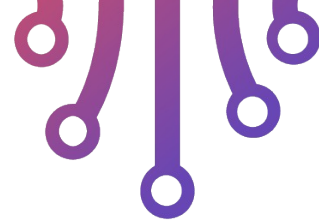




Versioning

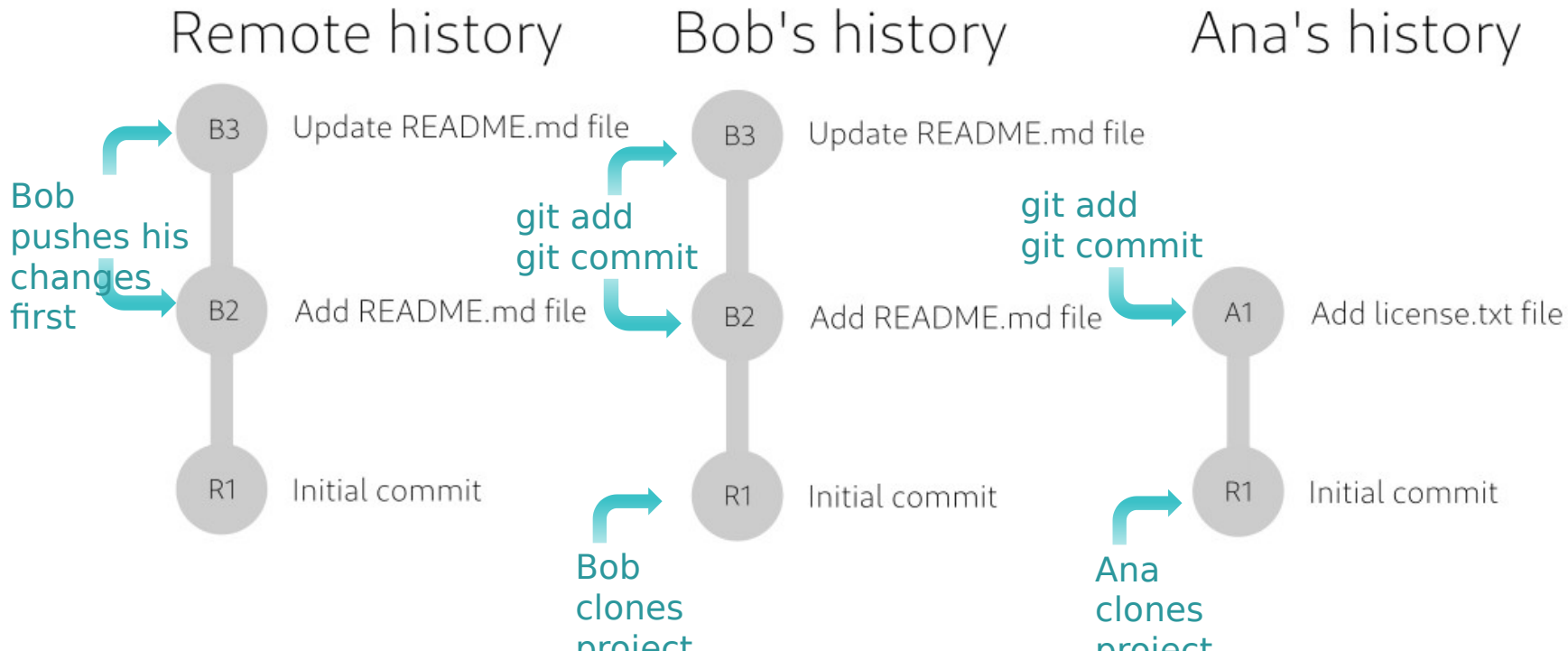
- Working example 2 devs (ana & bob)

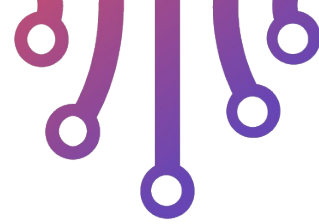




Versioning

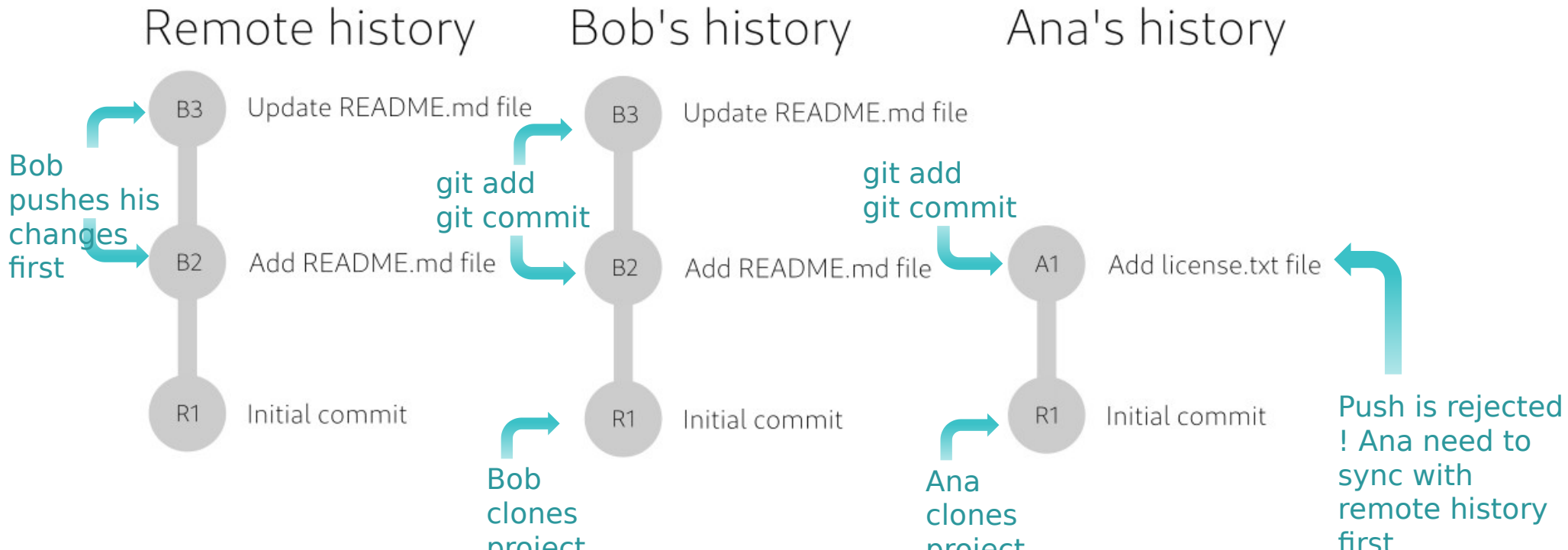
- Working example 2 devs (ana & bob)

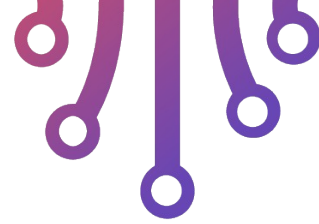




Versioning

- Working example 2 devs (ana & bob)





Versioning

- Working example 2 devs (ana & bob)

Ana execute command to pull remote changes

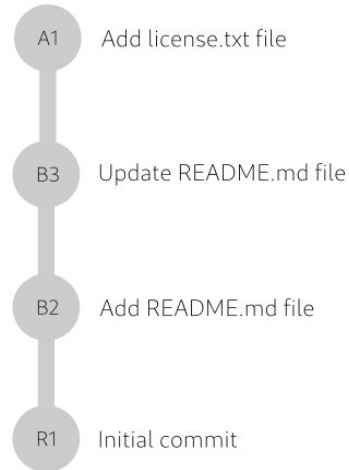
git pull

When pulling the differer

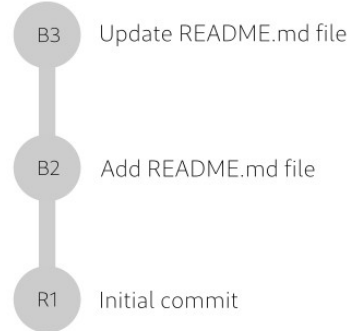
git push

Ana's commit goes after

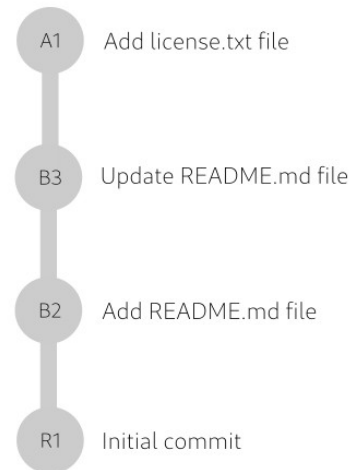
Remote history

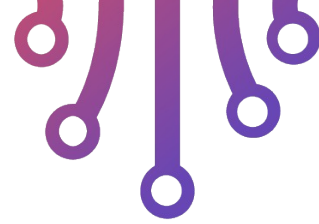


Bob's history



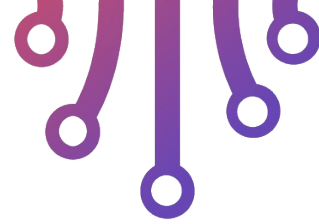
Ana's history





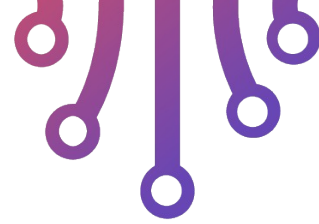
Versioning

- Conflicts can happen and will happen
- Always branch before start working
`git checkout -b "my-branch-name"`
- 2 devs work on the same file and same branch
Last dev to push, is the one that must solve the conflict
- Avoid commands that rewrite the history
Never use -> git rebase
- Avoid committing on the master branch
Create a new branch for each dev



Versioning

- Gitlab merge-request
- Every change on the main branch affect all users and the new release
- In gitlab a merge-request allows to keep track of changes on the main branch (simplification)
- Advantages on the CI process:
 - Keep a clear track of what changes are upcoming
 - Show the code changes for a specific request
 - Allows a review process to take place
 - Ideally not the same dev will look at the code and validate
 - Ideally not the same dev will merge the changes



Versioning

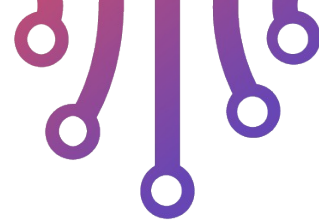
- Gitlab merge-request

nyax-tech > public > keystore_rest_server > Merge requests

Merge requests are a place to propose changes you've made to a project and discuss those changes with others

Interested parties can even contribute by pushing commits if they want to.

[New merge request](#)



Versioning

- Gitlab merge-request

nyax-tech > public > keystore_rest_server > Merge requests

Merge requests are a place to propose changes you've made to a project and discuss those changes with others

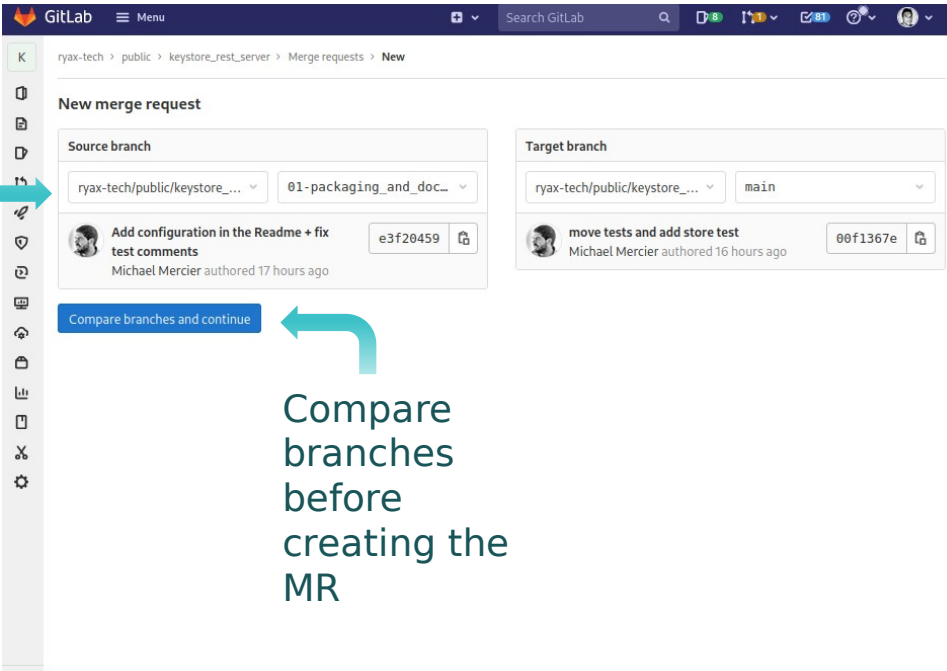
Interested parties can even contribute by pushing commits if they want to.

[New merge request](#)

Versioning

- Gitlab merge-request

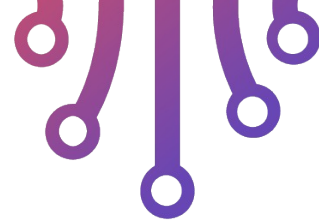
Choose a source branch



The screenshot shows the GitLab web interface for creating a new merge request. The page title is 'New merge request'. It features two main sections: 'Source branch' and 'Target branch'. The 'Source branch' section has a dropdown menu set to 'ryax-tech/public/keystore_...' and a selected branch '01-packaging_and_doc...'. Below this, a commit card is visible with the title 'Add configuration in the Readme + fix test comments', author 'Michael Mercier', and commit hash 'e3f20459'. The 'Target branch' section has a dropdown menu set to 'ryax-tech/public/keystore_...' and a selected branch 'main'. Below this, a commit card is visible with the title 'move tests and add store test', author 'Michael Mercier', and commit hash '00f1367e'. At the bottom of the form is a blue button labeled 'Compare branches and continue'. Three teal arrows point to these elements: one to the source branch dropdown, one to the target branch dropdown, and one to the 'Compare branches and continue' button.

Choose a target branch

Compare branches before creating the MR



Versioning

- Gitlab merge-request

The screenshot shows the GitLab web interface for creating a new merge request. The browser address bar indicates the repository path: `ryax-tech > public > keystore_rest_server > Merge requests > New`. The page title is "New merge request".

The interface is divided into two main sections: "Source branch" and "Target branch".

Source branch:

- Repository: `ryax-tech/public/keystore_...`
- Branch: `01-packaging_and_doc...`
- Commit: `e3f20459`
- Commit message: "Add configuration in the Readme + fix test comments"
- Author: Michael Mercier, authored 17 hours ago.

Target branch:

- Repository: `ryax-tech/public/keystore_...`
- Branch: `main`
- Commit: `00f1367e`
- Commit message: "move tests and add store test"
- Author: Michael Mercier, authored 16 hours ago.

A blue button labeled "Compare branches and continue" is located below the source branch information.

Versioning

- Gitlab merge-request

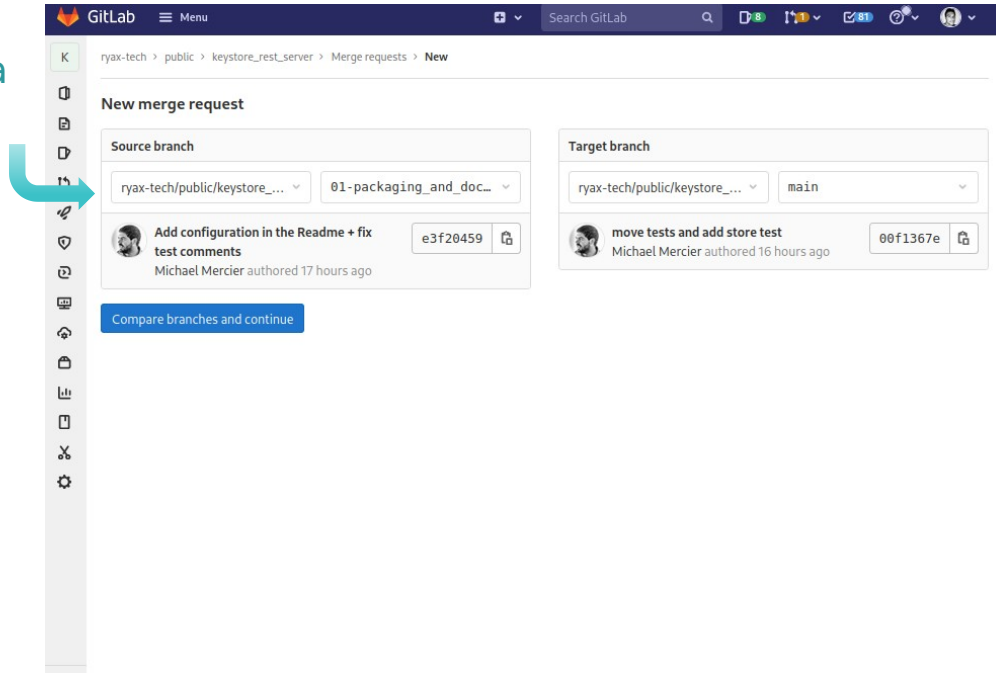
Choose a
source
branch

The screenshot shows the GitLab web interface for creating a new merge request. The page title is 'New merge request'. Under the 'Source branch' section, there are two dropdown menus: the first is set to 'ryax-tech/public/keystore_...' and the second is set to '01-packaging_and_doc...'. Below these, a commit card is displayed for the source branch: 'Add configuration in the Readme + fix test comments' by Michael Mercier, with commit hash 'e3f20459'. Under the 'Target branch' section, the dropdown is set to 'main'. Below this, a commit card is displayed for the target branch: 'move tests and add store test' by Michael Mercier, with commit hash '00f1367e'. A blue button labeled 'Compare branches and continue' is located at the bottom of the form. A teal arrow points from the text 'Choose a source branch' to the first dropdown menu in the 'Source branch' section.

Versioning

- Gitlab merge-request

Choose a source branch



Choose a target branch

Versioning

- Gitlab merge-request

Choose a source branch



ryax-tech > public > keystore_rest_server > Merge requests > New

New merge request

Source branch

ryax-tech/public/keystore_... 01-packaging_and_doc...

Target branch

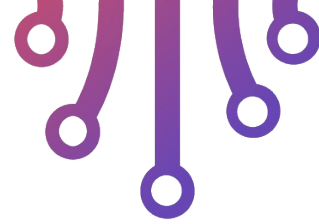
ryax-tech/public/keystore_... main

Compare branches and continue

Compare branches before creating the MR

Choose a target branch





Versioning

- Gitlab merge-request
 - Set a title and description
 - Assignees
 - Person that repondes to t
 - Reviewers
 - Person to approve this M

GitLab Menu

ryax-tech > public > keystore_rest_server > Merge requests > New

New merge request

From 01-packaging_and_docker-solution into main [Change branches](#)

Title
[Start the title with Draft:](#) to prevent a merge request that is a work in progress.
[Add description templates](#) to help your contributors communicate effectively.

Description [Write](#) [Preview](#)

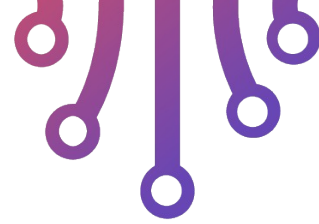
[Markdown and quick actions](#) are supported.

Assignees [Assign to me](#)

Reviewers
1 member must approve to merge. Anyone with role Developer or higher.
[Approval rules](#)

Milestone

Labels



Versioning

- Gitlab merge-request
 - Scrolling down we can see
 - Changes
 - Commits

GitLab Merge Request interface for the project `keystore_rest_server`. The interface shows the 'Changes' tab with 2 changed files. The first file is `README.md`, which has 13 additions and 2 deletions. The diff shows a new section for configuration variables, including `REDIS_HOST`, `REDIS_PORT`, `REDIS_PASSWORD`, and `DEBUG`.

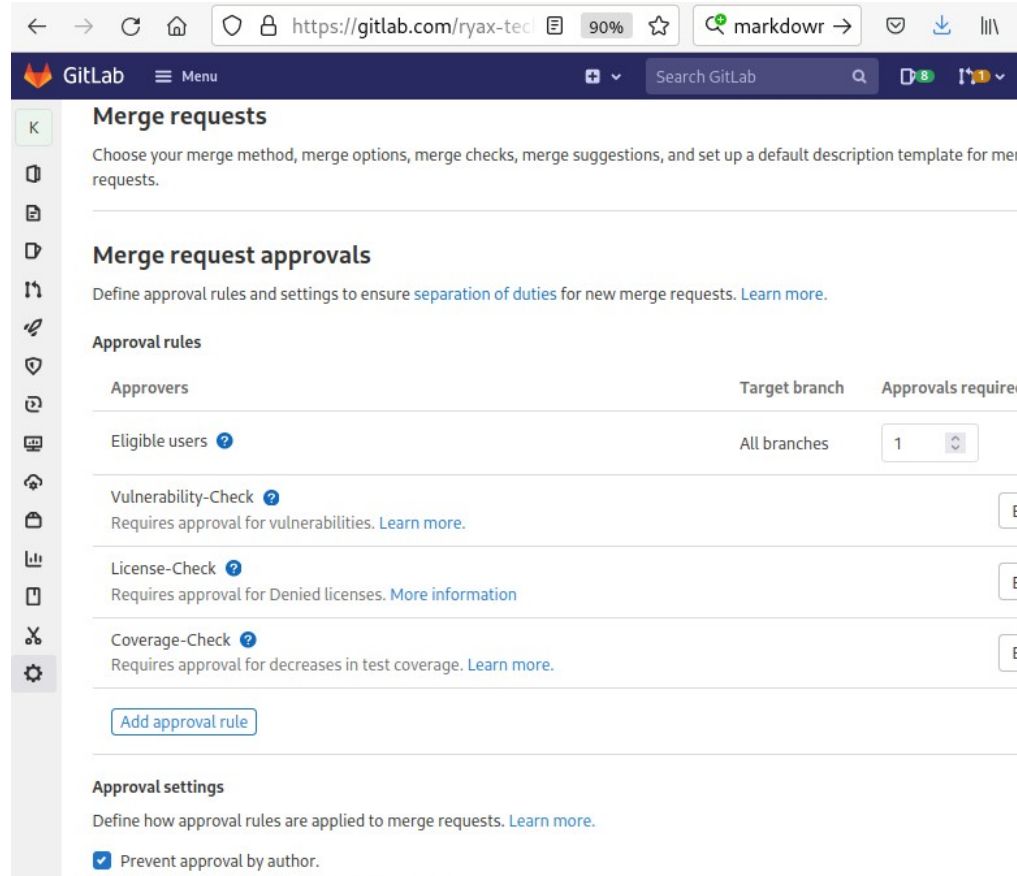
```
... @@ -2,3 +2,14 @@
2 2
3 3 This is a simple Python web server example application. It provides a Keystore REST
4 4 API based on a Redis storage.
5 +
6 + ## Configuration
7 +
8 + The API is serving by default on the port 5000.
9 +
10 + Configuration variables are:
11 + - REDIS_HOST: the redis server host (default: redis)
12 + - REDIS_PORT: the redis server port (default: 6379)
13 + - REDIS_PASSWORD: the redis server password (default: password)
14 + - DEBUG: Run the Flask web server in Debug mode is set to "true" (which is the
15 + default)
```

The second file is `test_test_end2end.py`, which has 7 additions and 3 deletions. The diff shows a new test function `test_create_read()` with curl commands and a requests.post() call.

```
... @@ -3,7 +3,7 @@ import requests
3 3
4 4 def test_create_read():
5 5     # create
6 - # curl -X POST http://127.0.0.1:5000/create -H 'Content-Type: application-json' -d '{"key": "d
7 + # curl -X POST http://127.0.0.1:5000/keystore -H 'Content-Type: application-json' -d '{"key":
8     r = requests.post(
9         "http://localhost:5000/keystore", data={'key': "1", "value": "something"}
10    )
... @@ -16,7 +16,7 @@ def test_create_read():
```

Versioning

- Settings -> General
- Merge request approvals
 - # reviewers that must approve

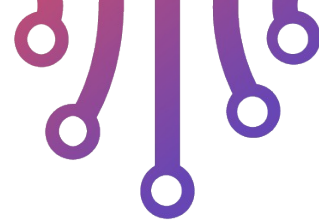


The screenshot shows the GitLab web interface for configuring Merge Request Approvals. The page title is "Merge requests" and the sub-section is "Merge request approvals". The main content area is titled "Approval rules" and contains a table with the following columns: "Approvers", "Target branch", and "Approvals required".

Approvers	Target branch	Approvals required
Eligible users ?	All branches	1 <input type="text"/>
Vulnerability-Check ? Requires approval for vulnerabilities. Learn more.		<input type="text"/>
License-Check ? Requires approval for Denied licenses. More information		<input type="text"/>
Coverage-Check ? Requires approval for decreases in test coverage. Learn more.		<input type="text"/>

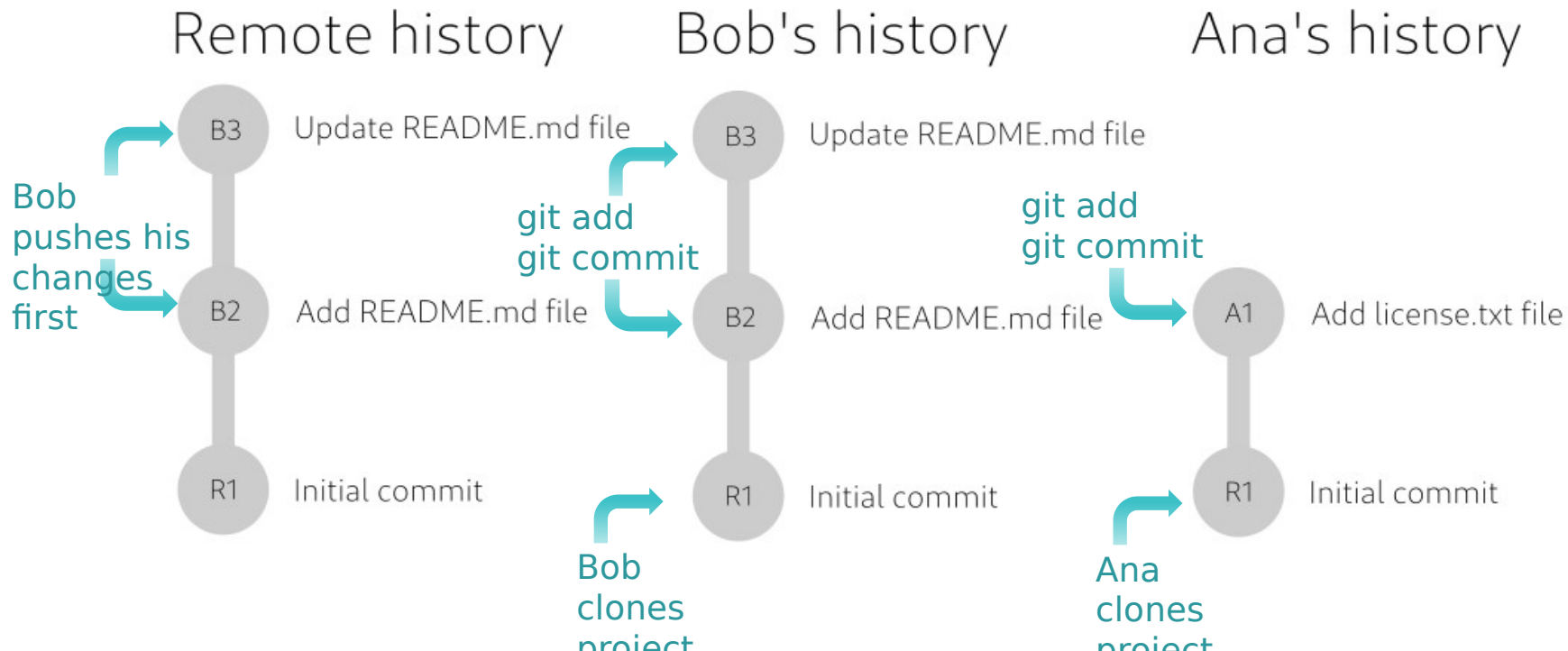
Below the table is a button labeled "Add approval rule".

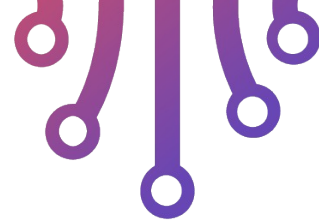
At the bottom of the page, under the "Approval settings" section, there is a checkbox labeled "Prevent approval by author." which is checked.



Versioning

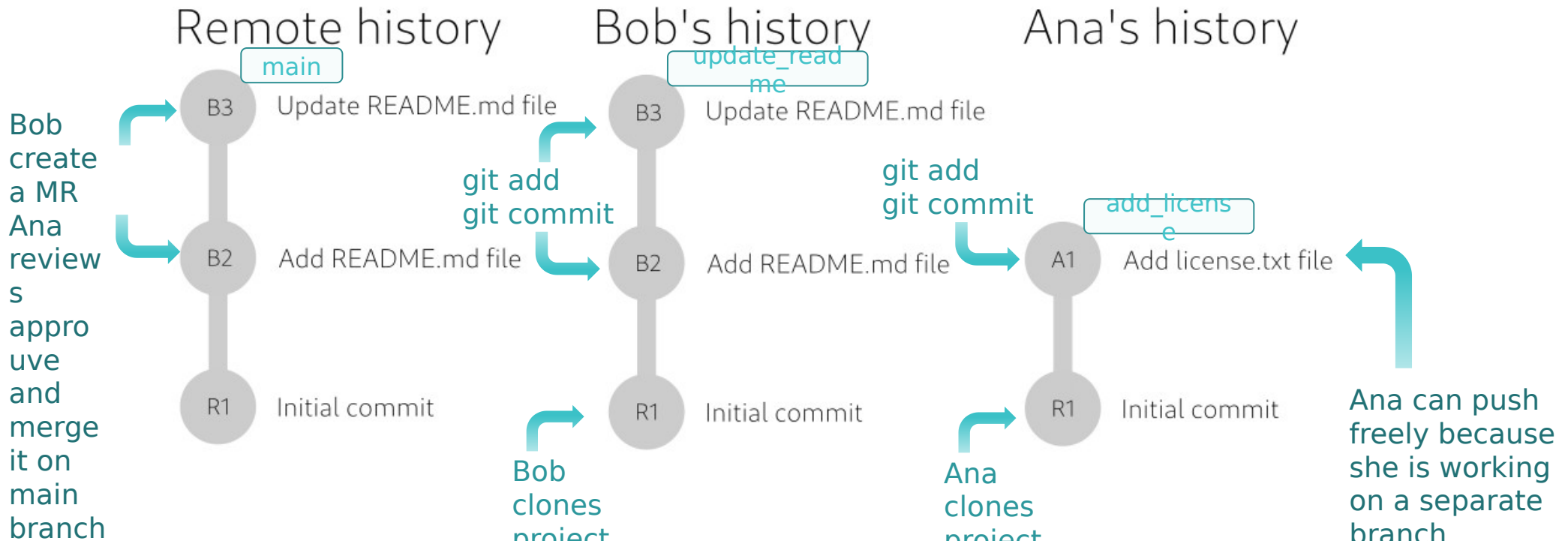
- How this scenario would work with gitlab review?

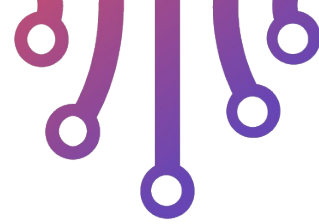




Versioning

- How this scenario would work with gitlab review?





Versioning

- Work on a multi-tenant gitlab project
 - Precommit - add basic coding checks style
 - merge request
 - review process
 - approval system