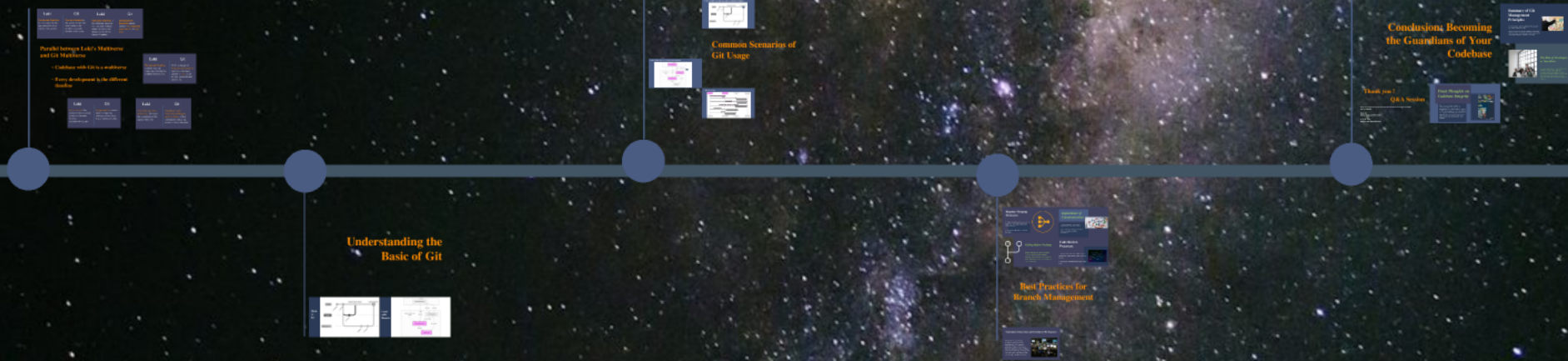


Be the Cosmic Guardian of Your Codebase with Git

Navigating Git Branches with



PHUSE EU Connect 2024 – OS15

Author: Shuai Wu, Brian M. Lang, Carl Herremans,
Christian Koul, MSD

Parallel between Loki's Multiverse and Git Multiverse

- Codebase with Git is a multiverse
- Every development is the different timeline

Loki

The Sacred Timeline

in Loki's world is the sequence of events that had no variant.

Git

The main branch is the series of commits which lead to the production-ready version of the code.

Loki

Alternate Timeline is the different possible realities that emerge when actions cause deviations from the Sacred Timeline.

Git

Development branches which contain **new features**, **experiments**, or **bug fixes**.

Loki

The Sacred Timeline

could be easily in chaos impacted by the smallest disturbance .

Git

With the design of **local/remote repostory**, conflicts and chaos caused in **local** could be easily avoided and controlled.

Loki

TVA (Time Variance Authority): Serves as the custodian of the Sacred Timeline.

Git

Developer team including developers and reviewers of the codebase to develop, maintain the codebase.

Loki

Nexus Events: The pivotal moments when a timeline deviates from its predetermined path.

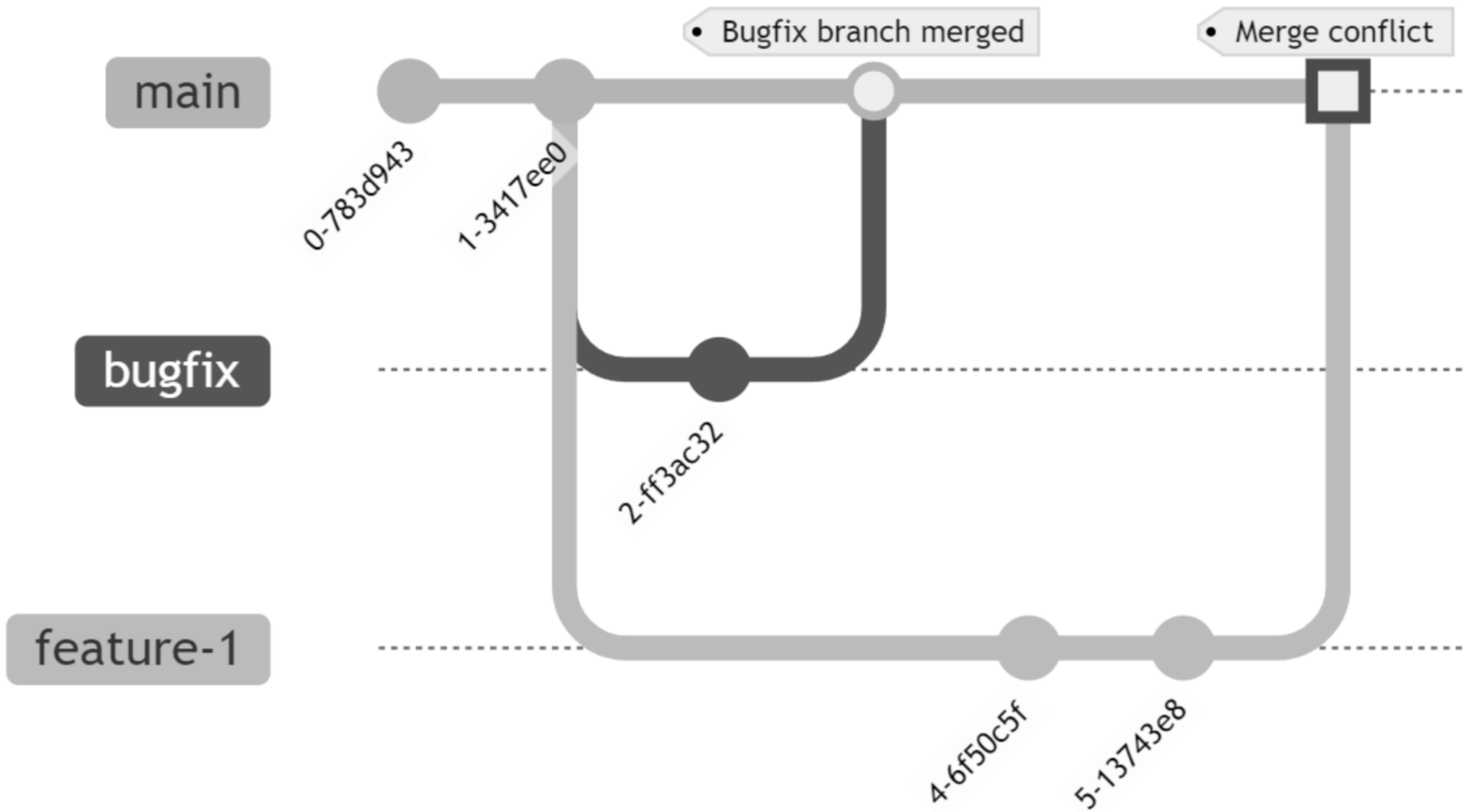
Git

Merge conflicts occurs when competing changes are made to the same line of a file.

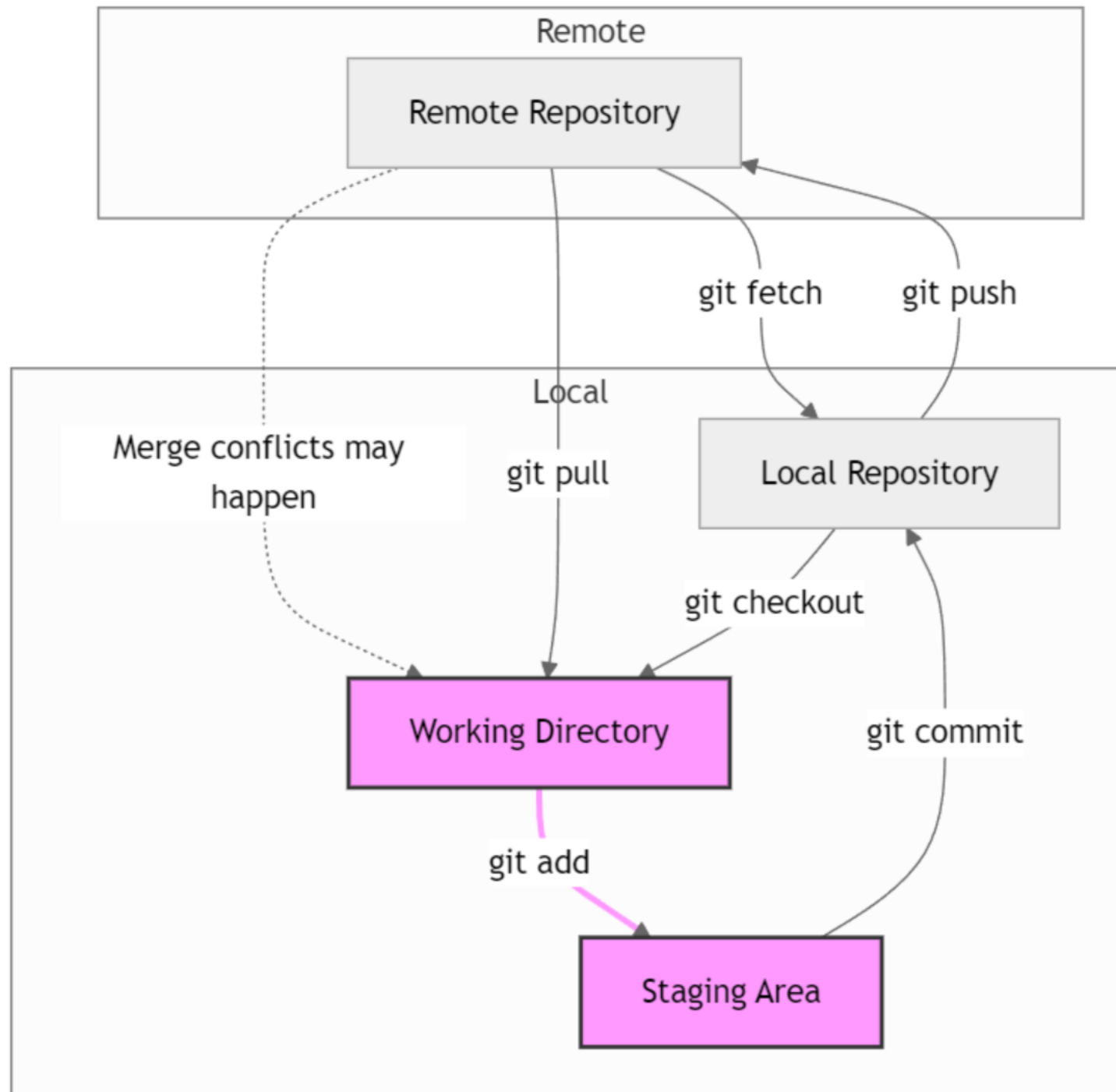


Understanding the Basic of Git

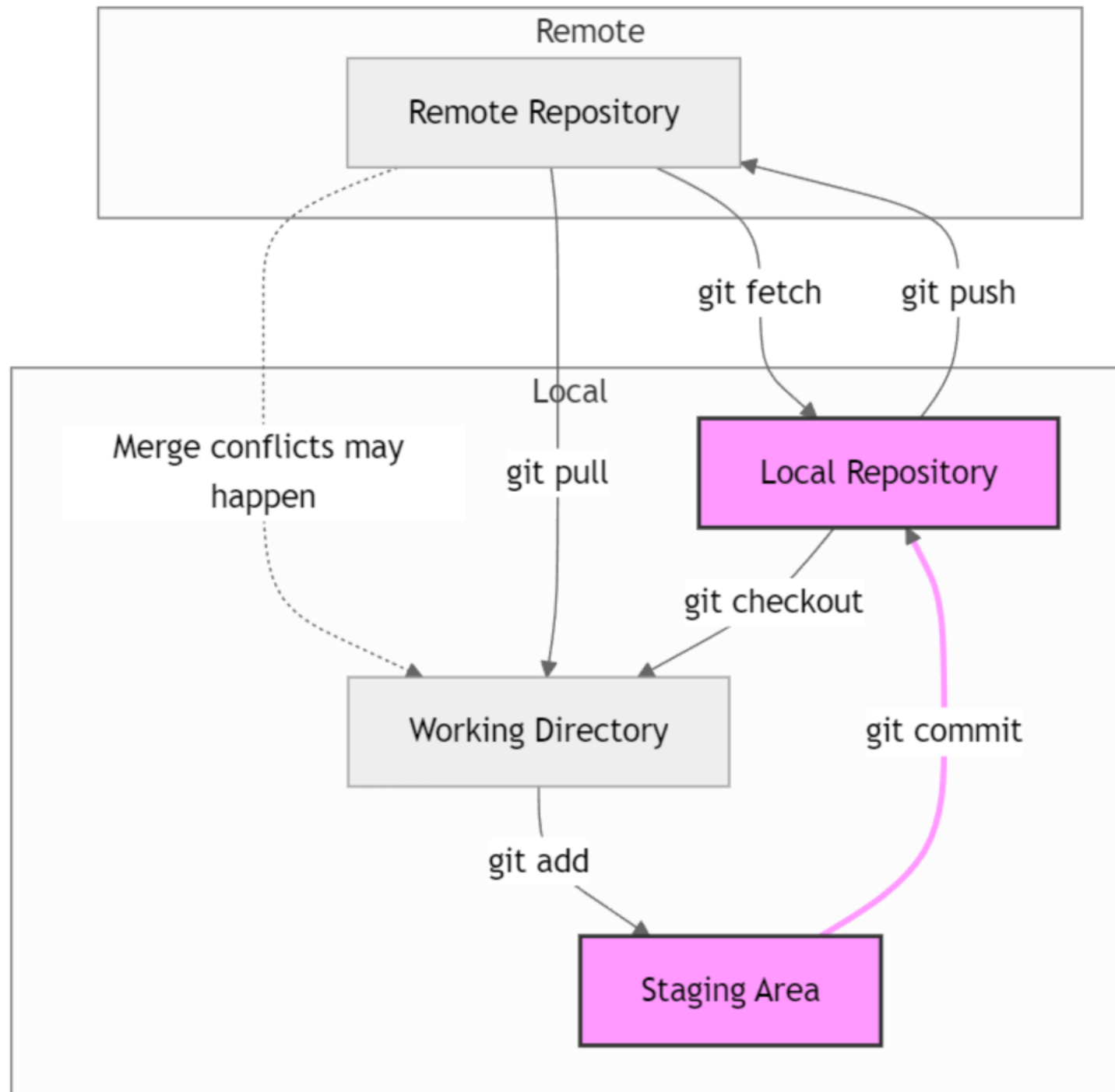
Main vs dev



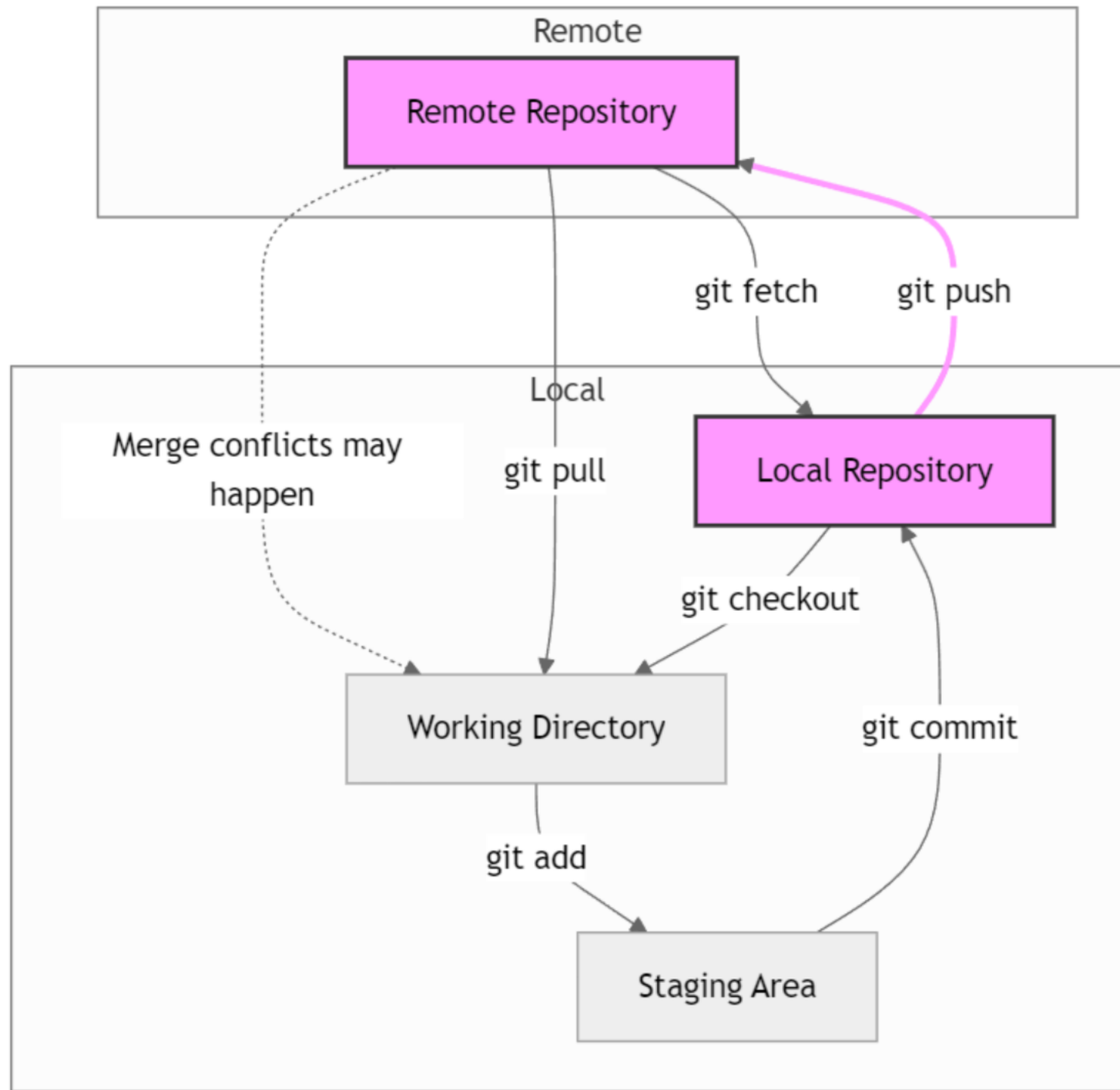
Local with Remote



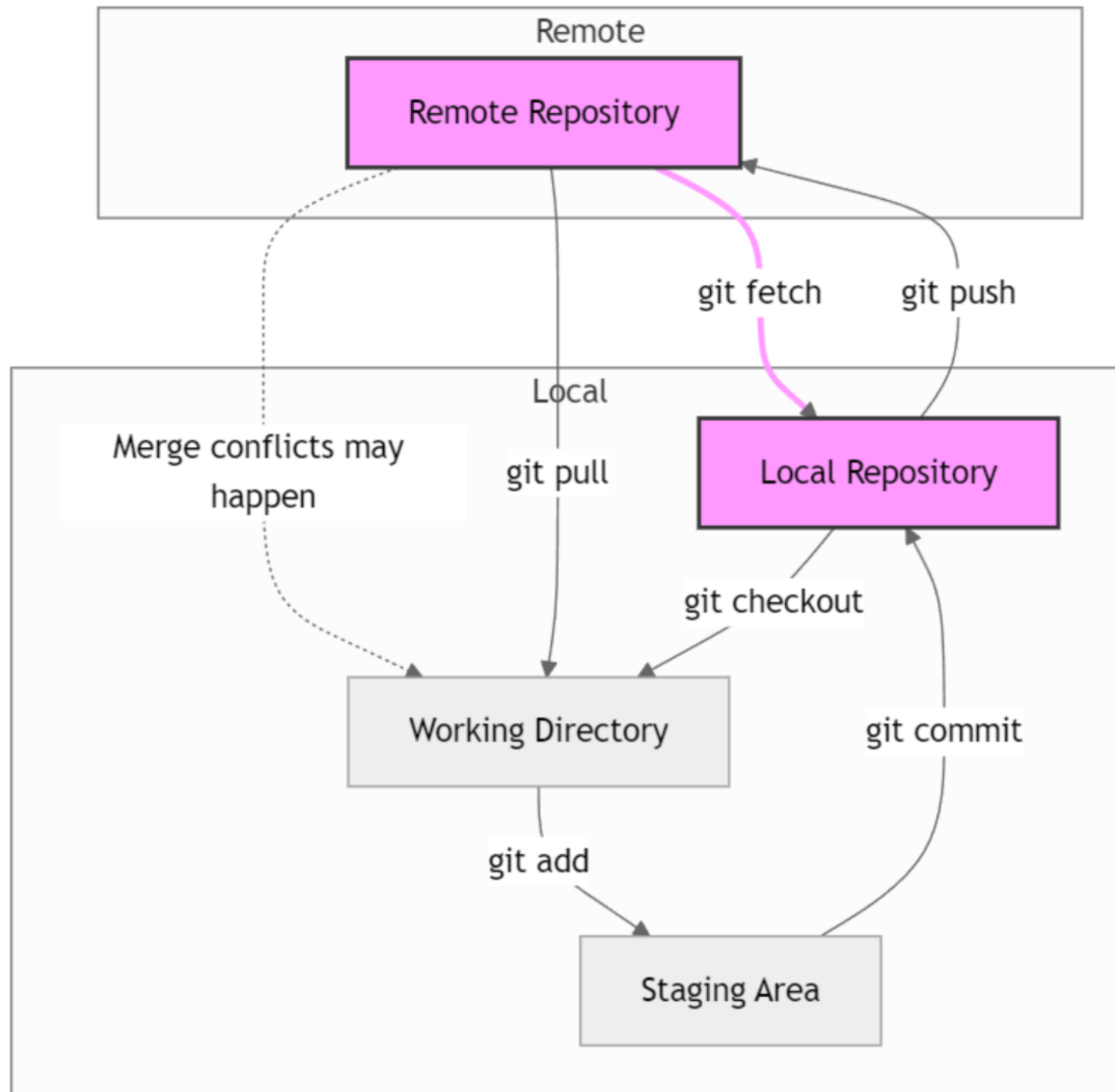
Local with Remote



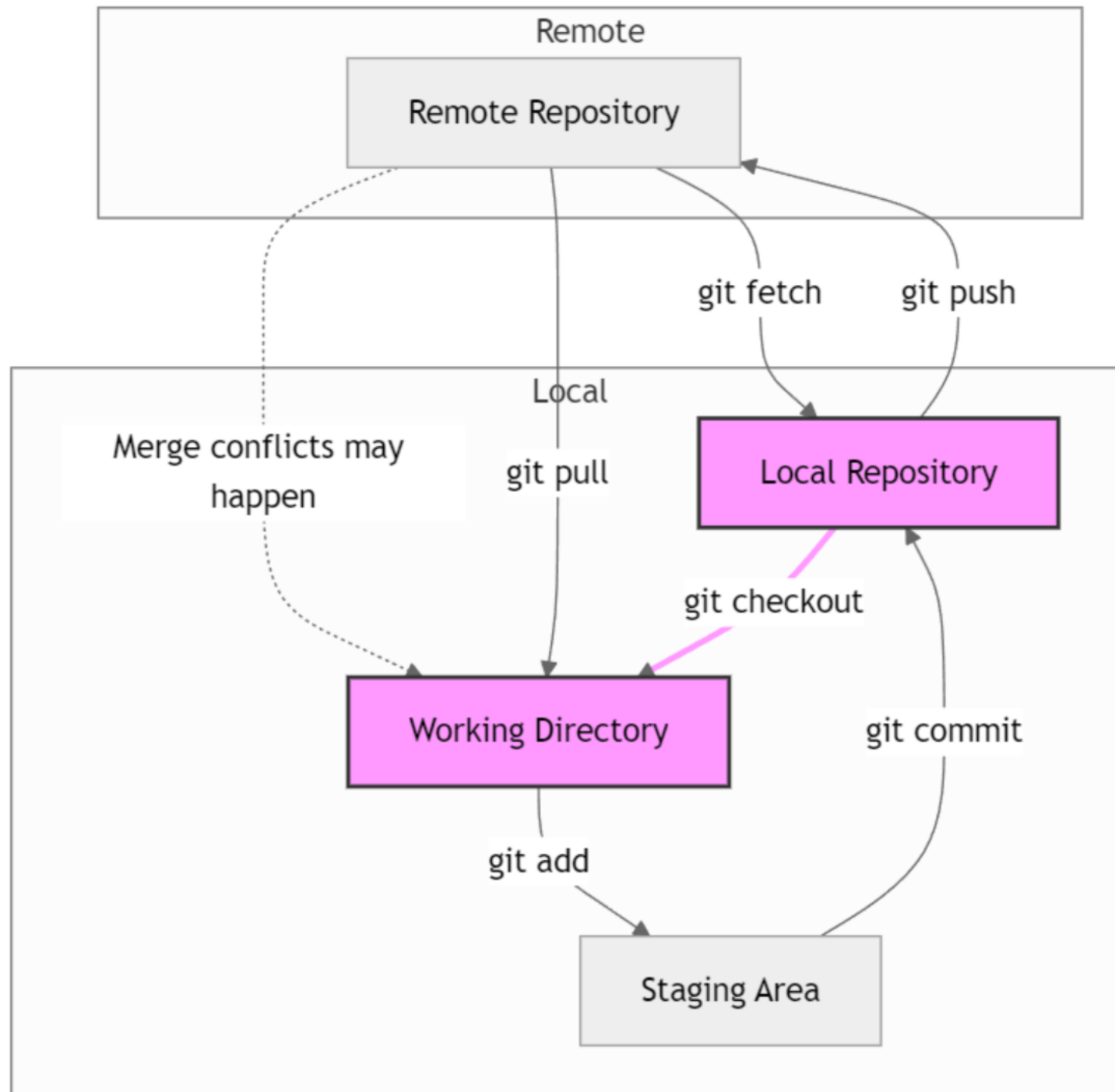
Local with Remote



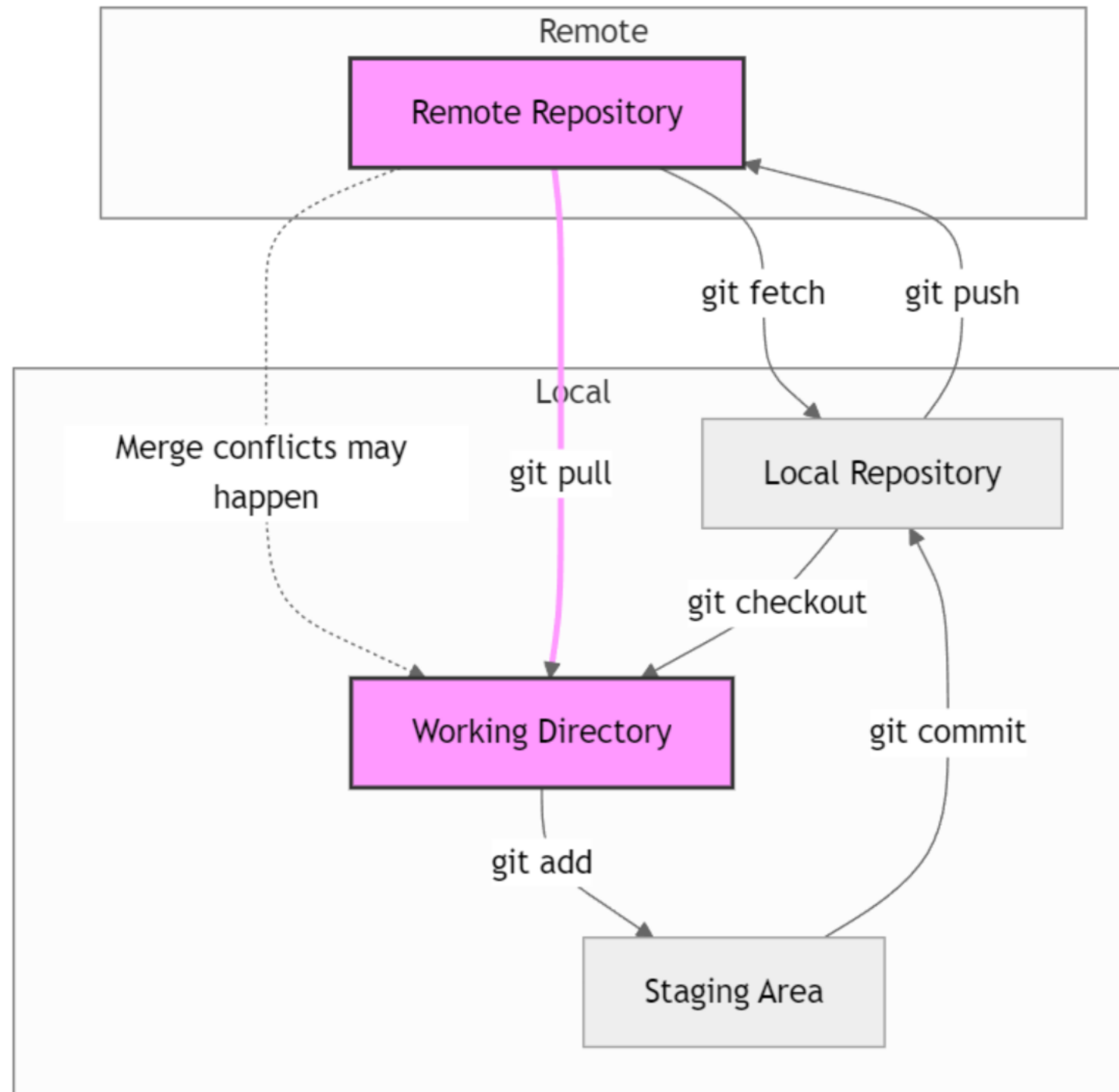
Local with Remote



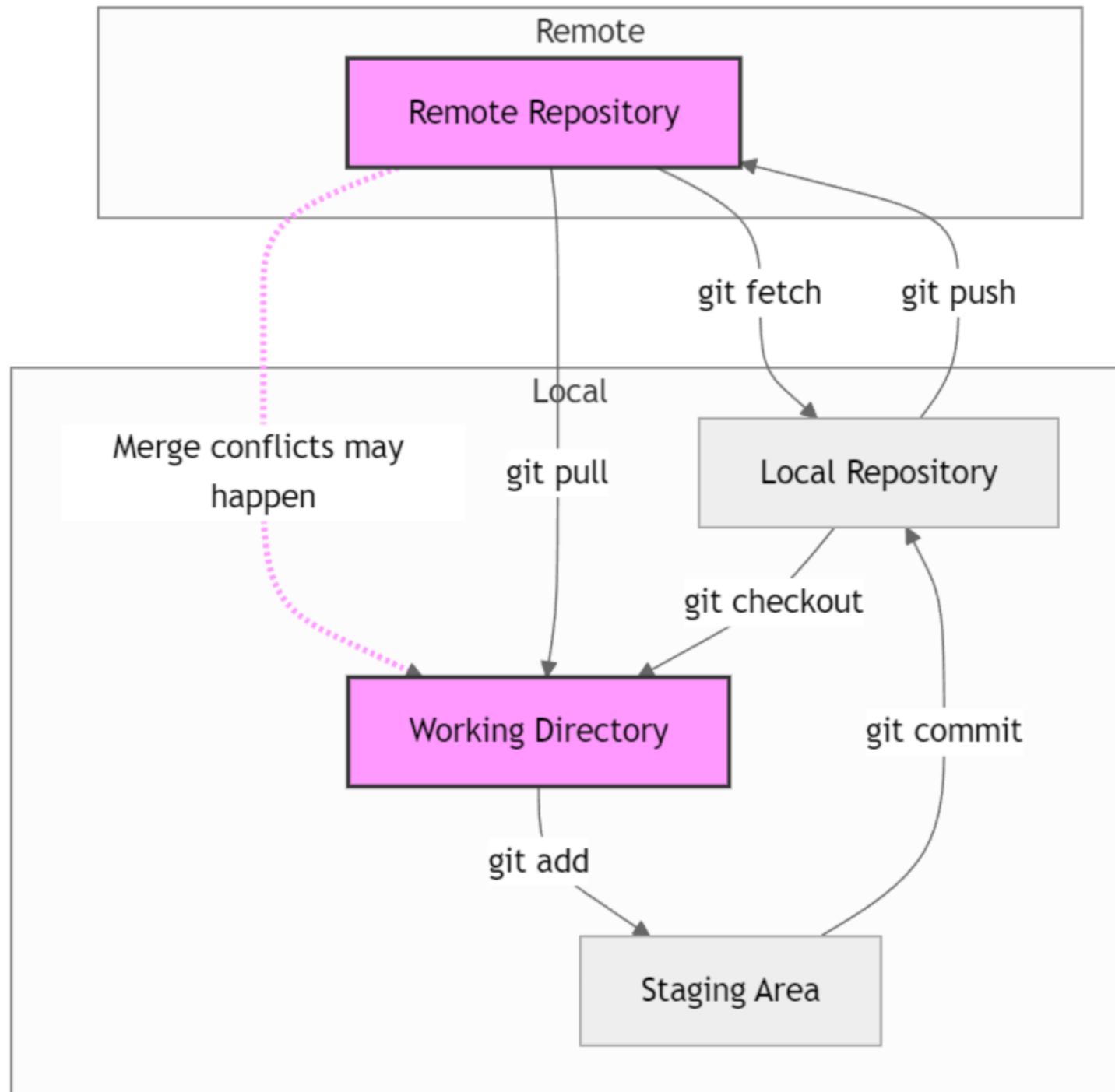
Local with Remote



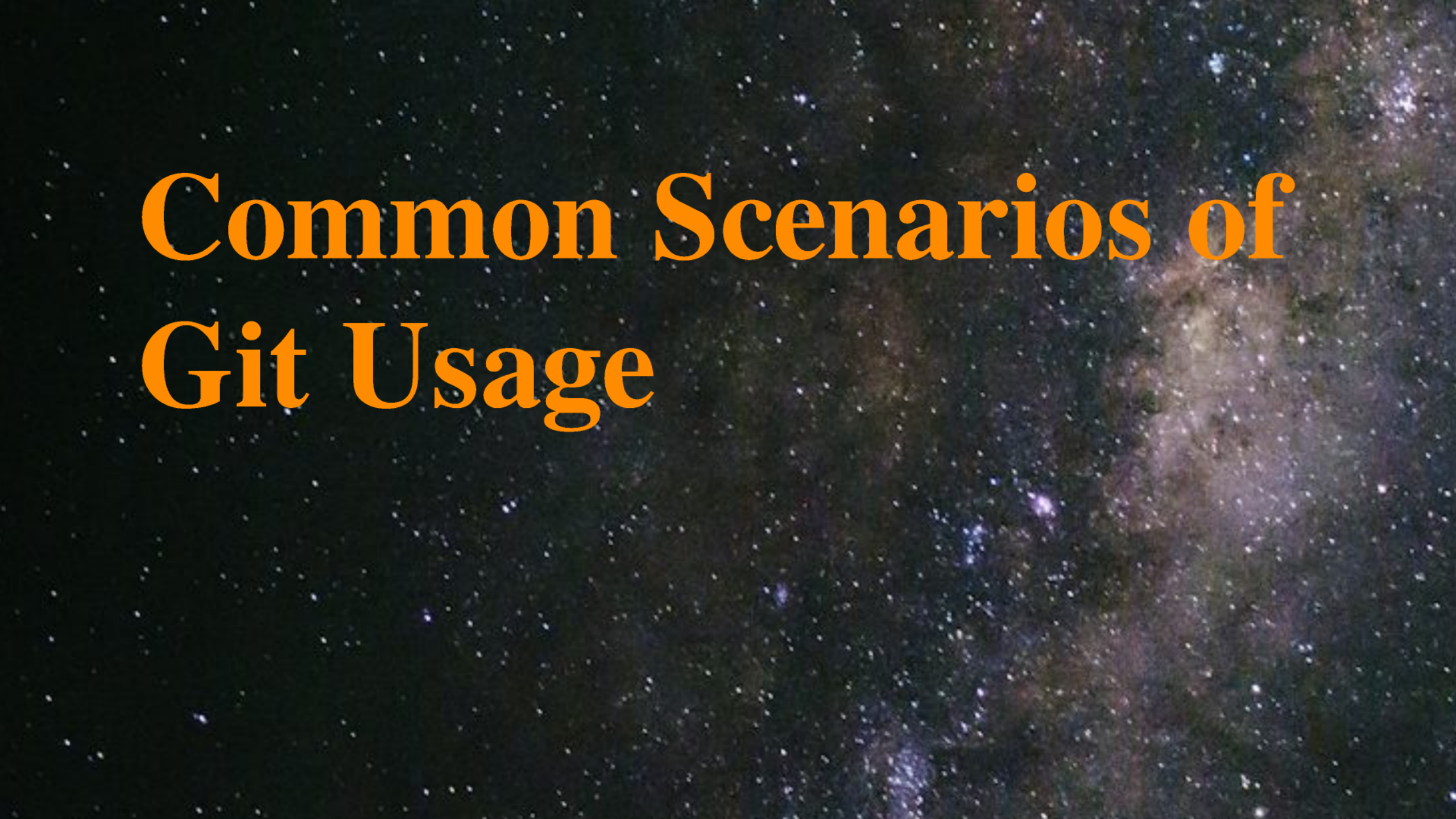
Local with Remote



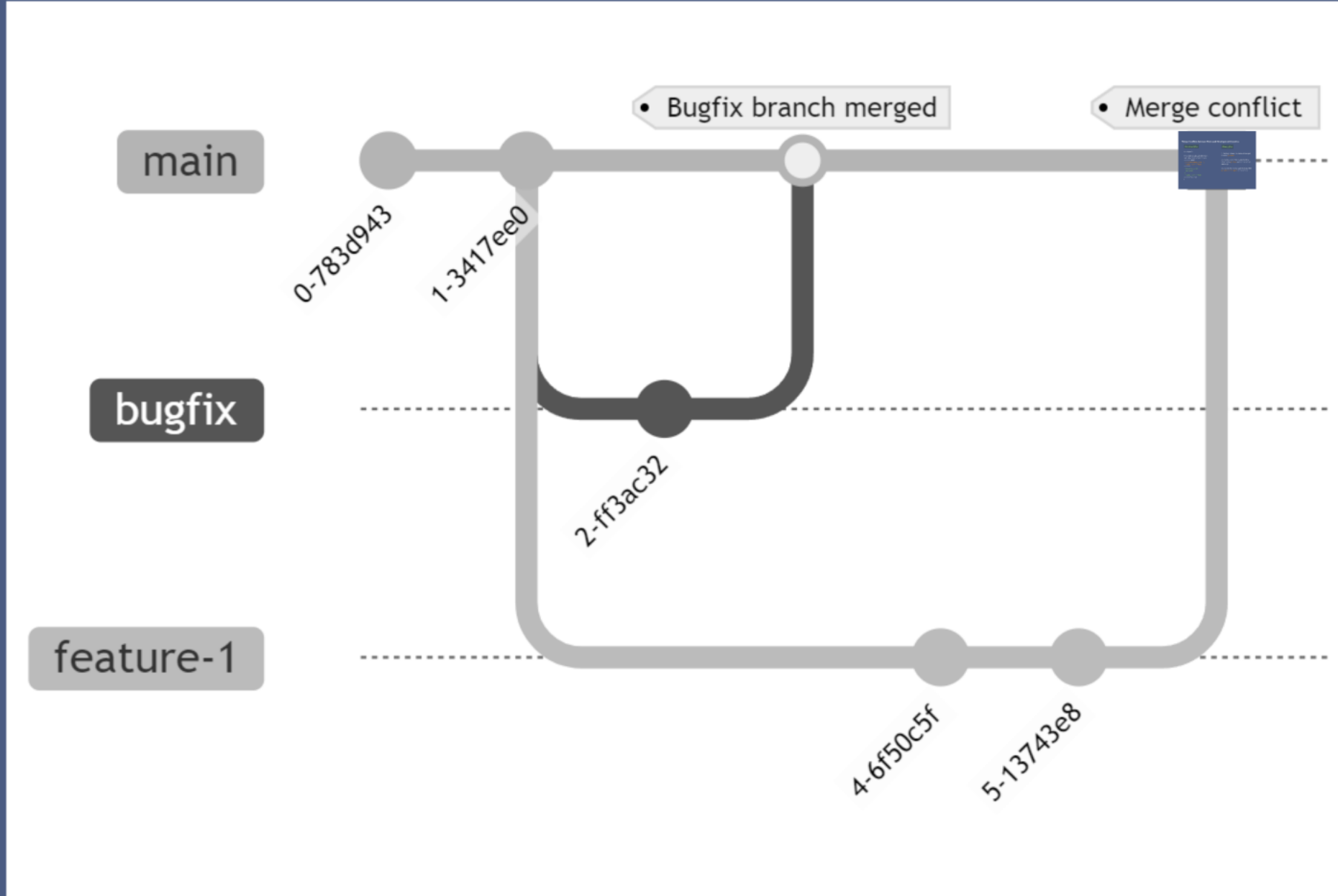
Local with Remote



Common Scenarios of Git Usage



Merge Conflicts between Main and Development Branches



Merge Conflicts between Main and Development branches

Text-based files:

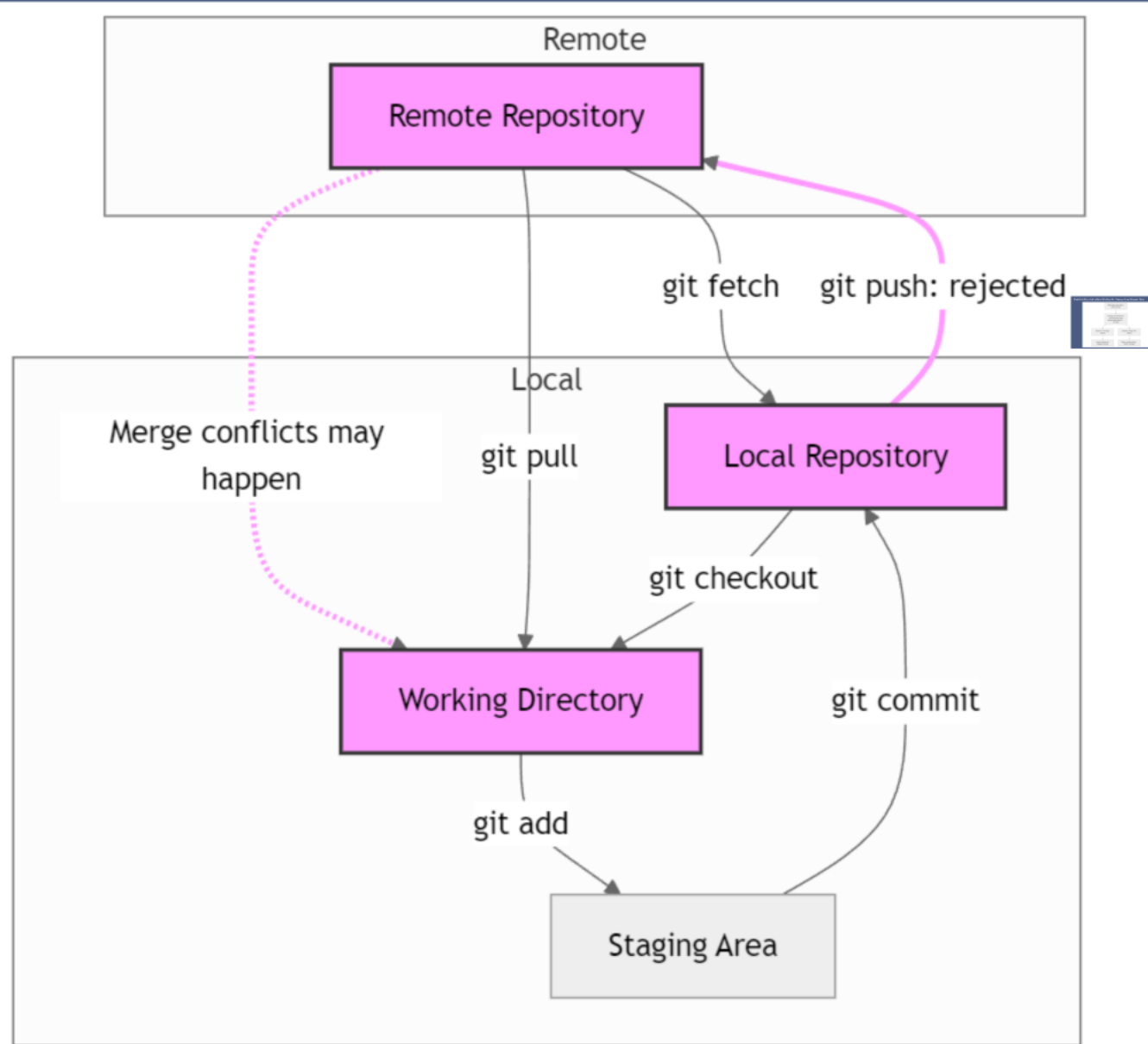
```
# analysis.R

# Function to calculate the mean
calculate_mean <- function(x) {
<<<<<<< HEAD
  print("Calculating mean...")
  mean(x, na.rm = TRUE)
=====
  if (length(x) == 0) {
    return(NA)
  }
  mean(x, na.rm = TRUE)
>>>>>>> feature-1
}
```

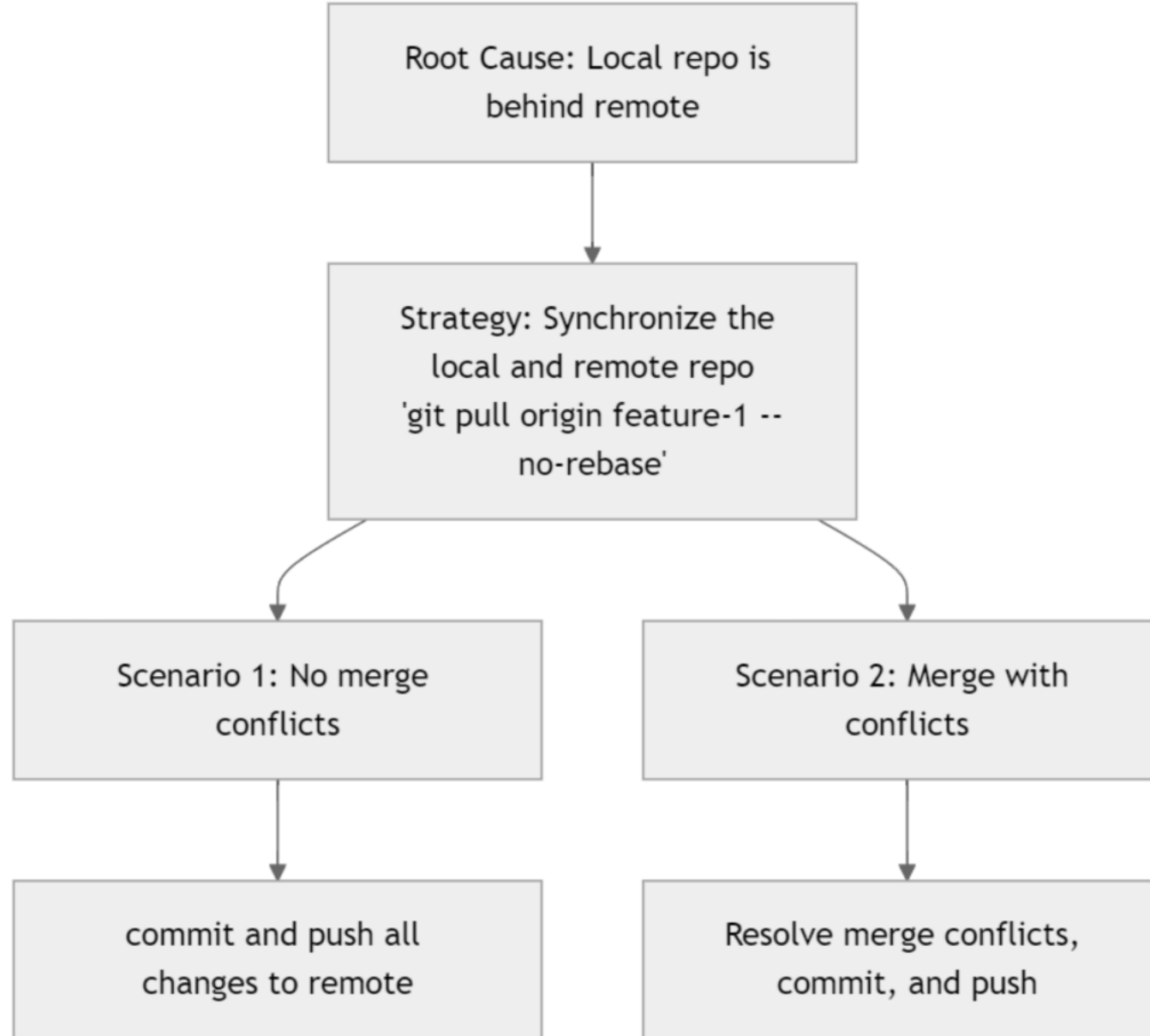
Binary files:

1. Getting to know the status of merged file with **git status**.
2. Checkout both files to your working directory by **git show** and compare them manually.
3. Choose the one you want to keep, then **git add, commit, push** to the remote.

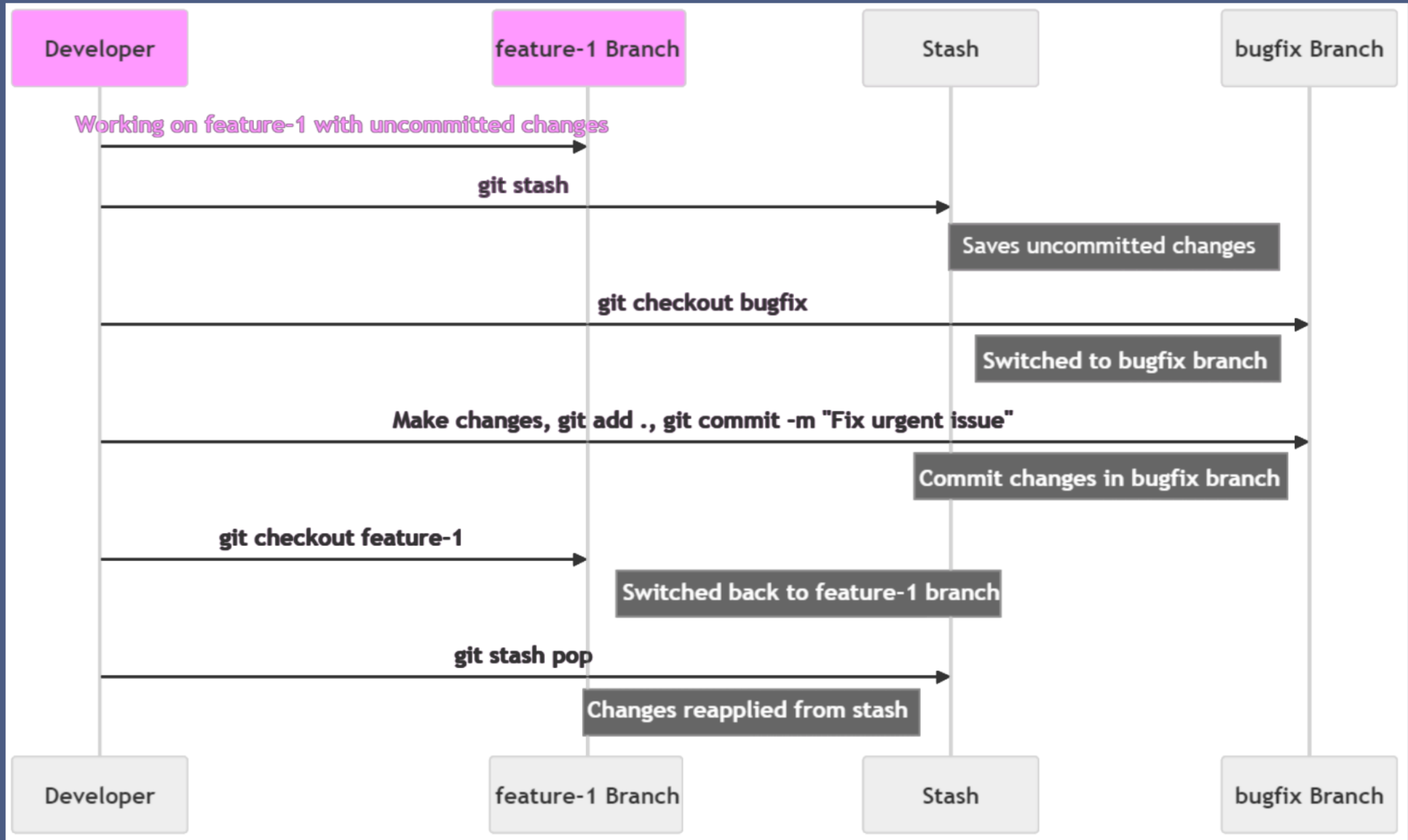
Merge Conflicts between Local and Remote Branches



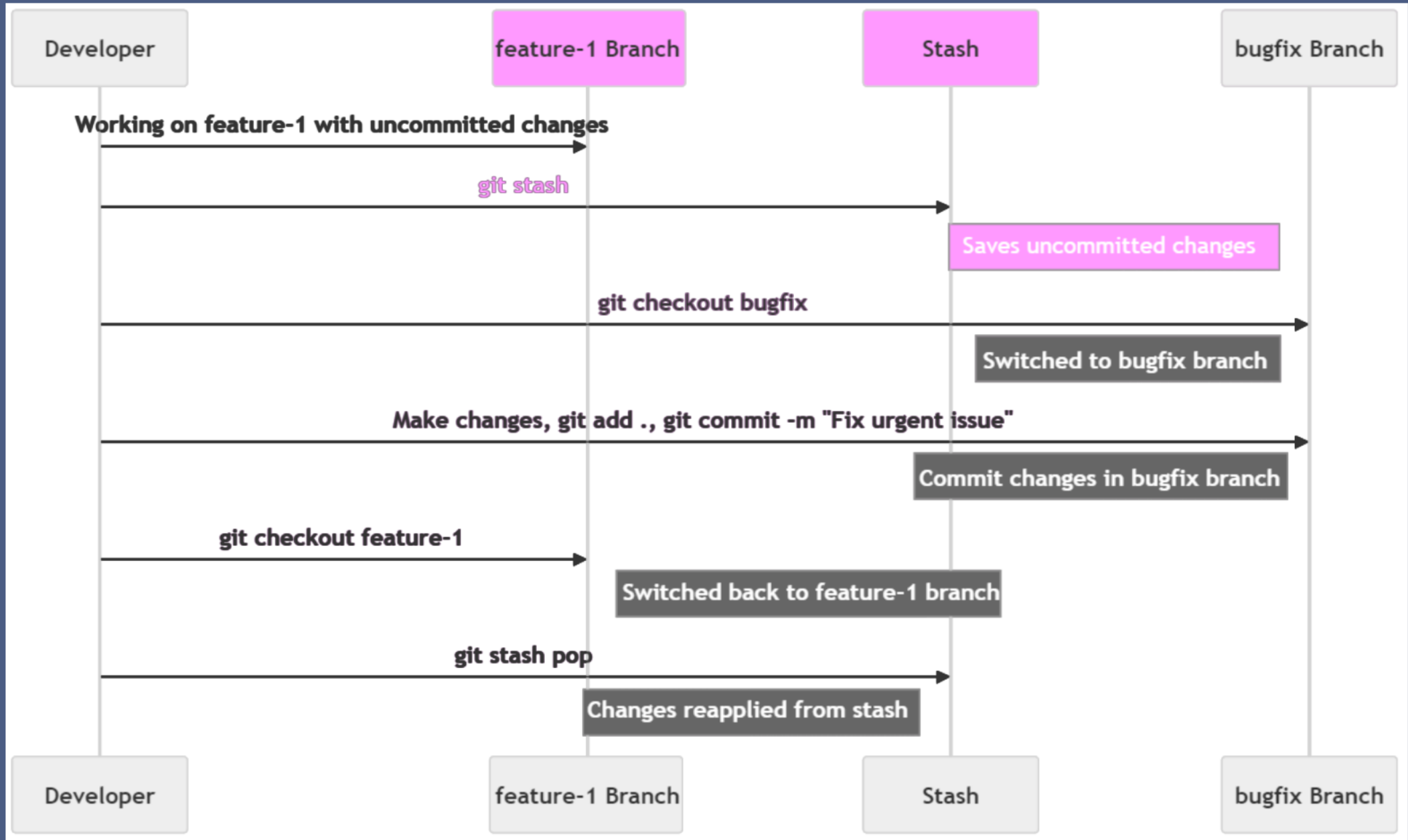
Push Got Rejected without Pulling the Changes from Remote First



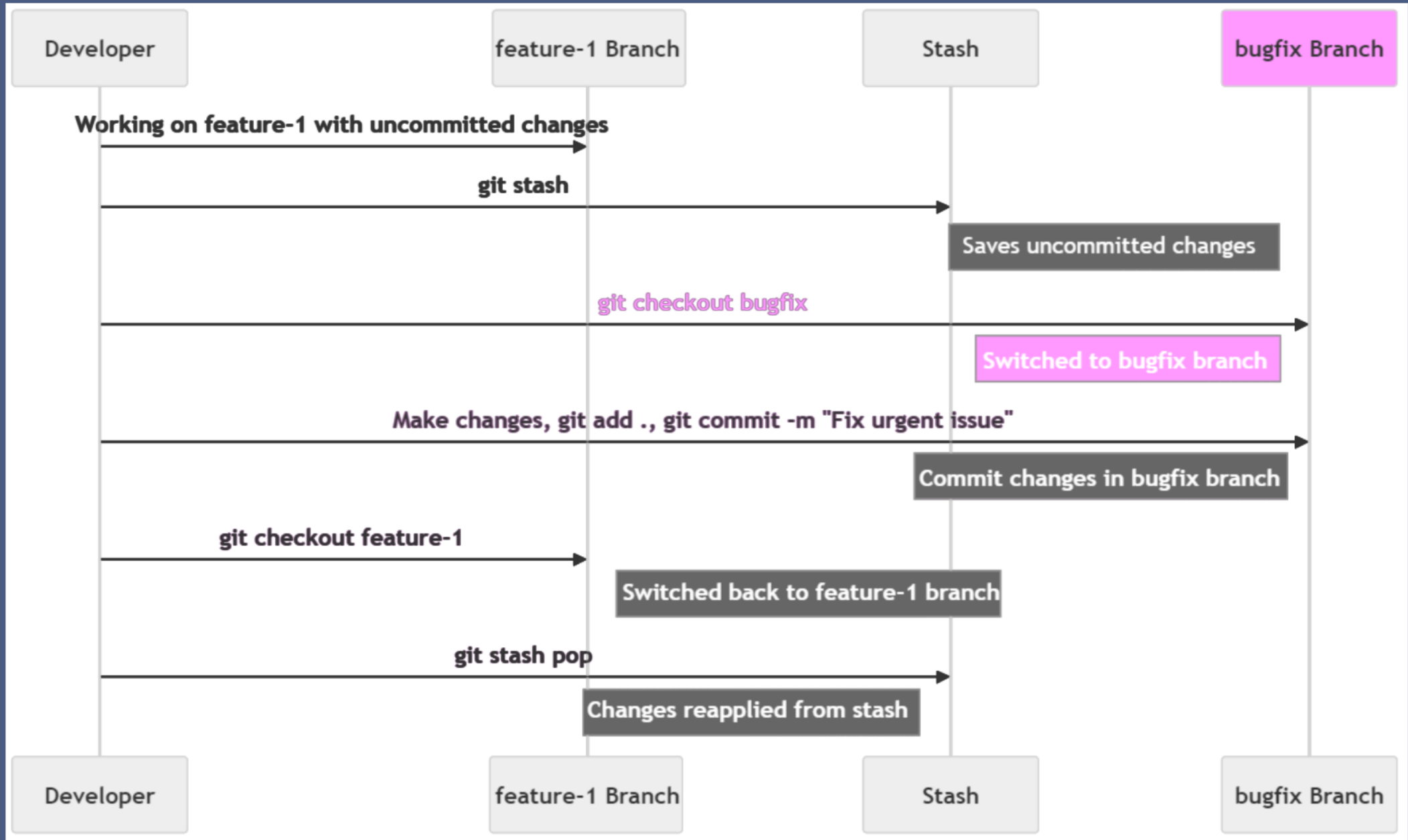
Stash the changes



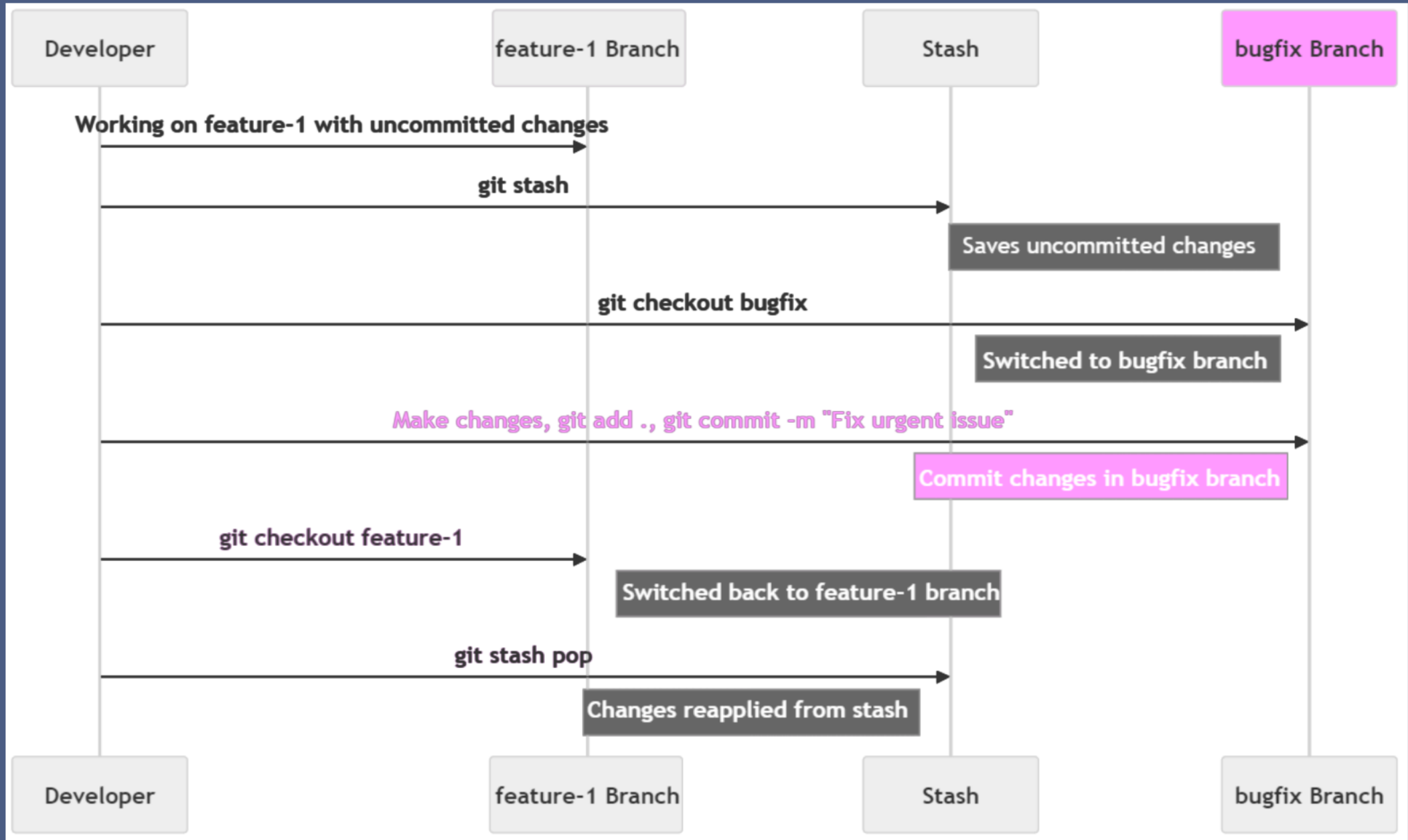
Stash the changes



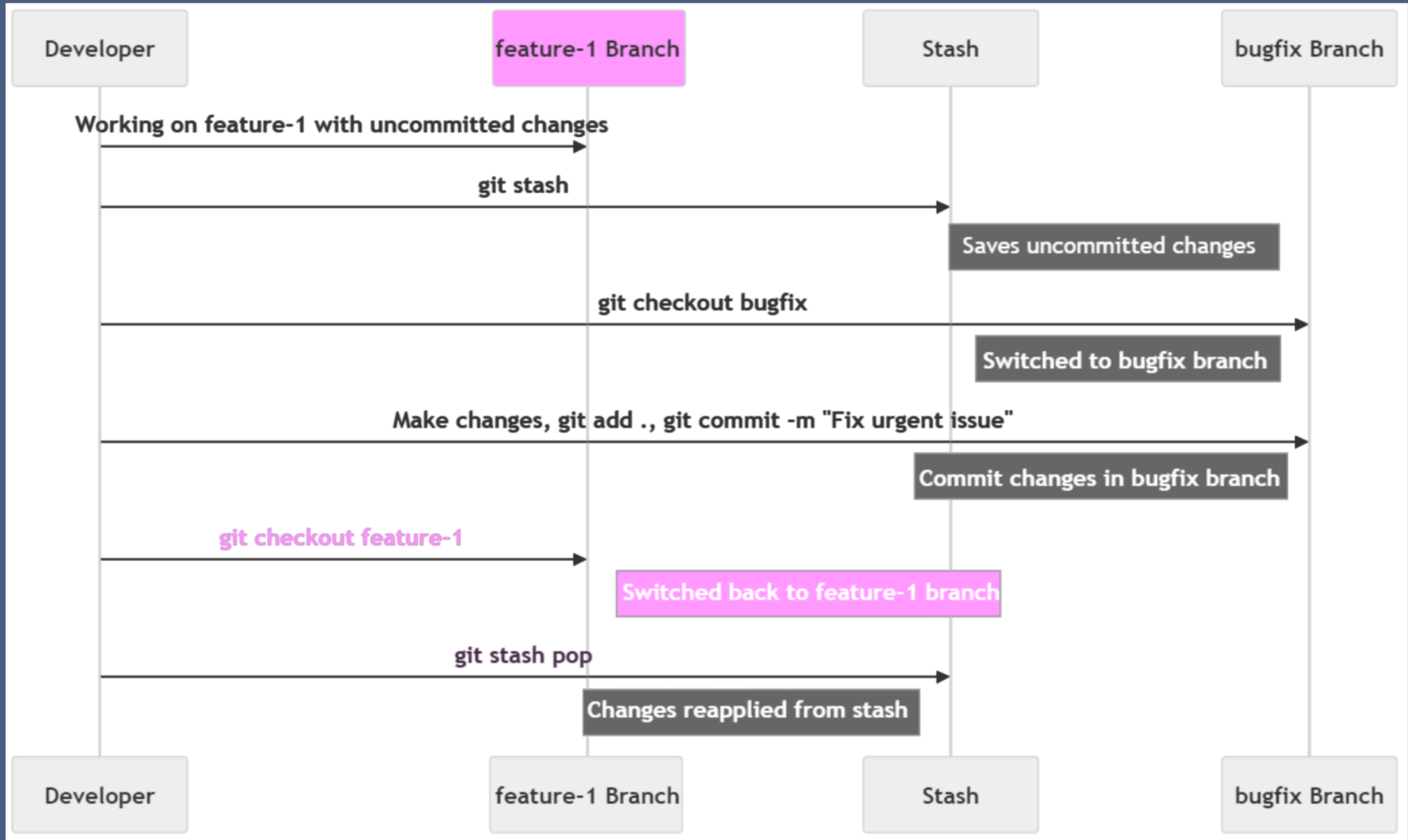
Stash the changes



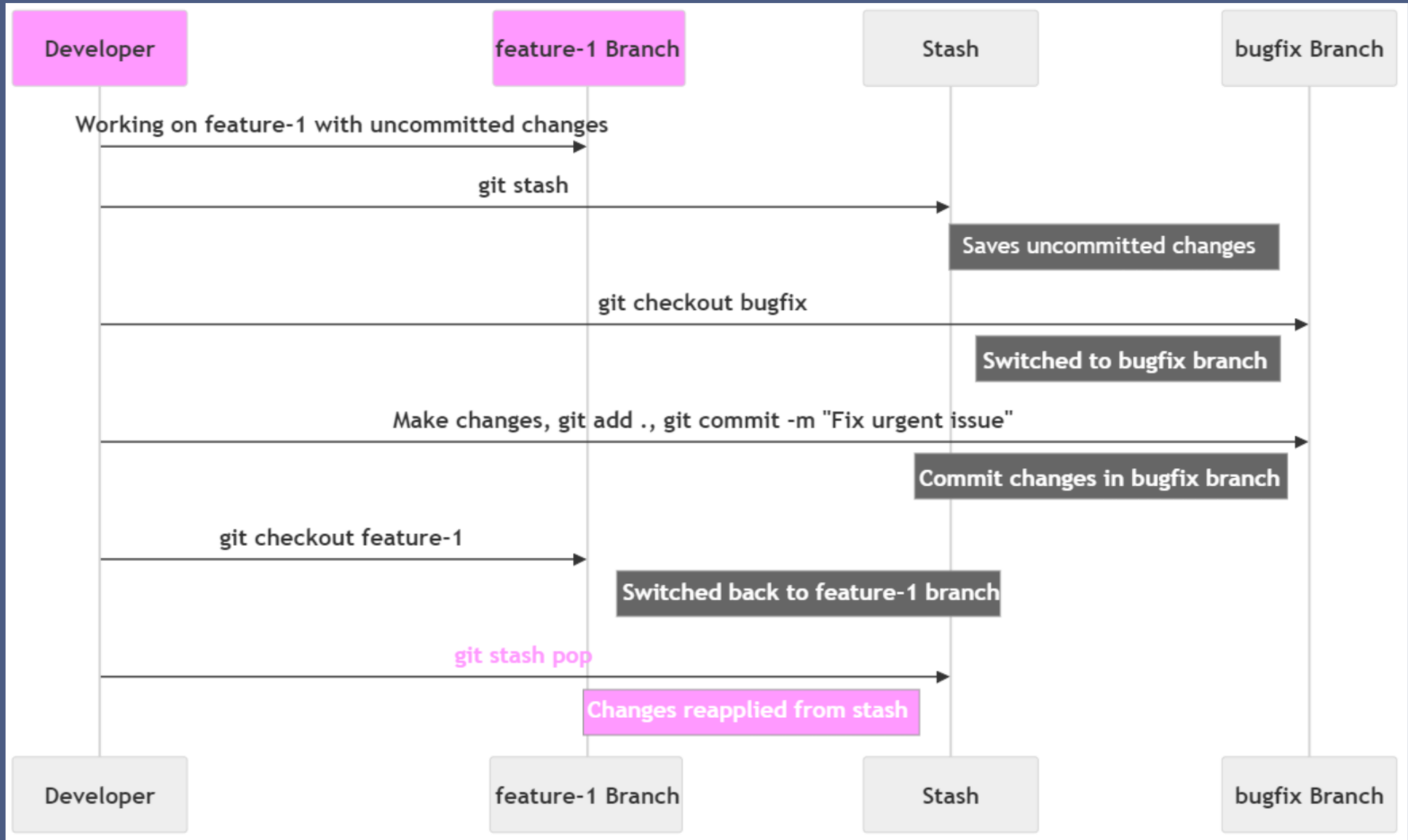
Stash the changes



Stash the changes



Stash the changes





Best Practices for Branch Management

Importance of Communication

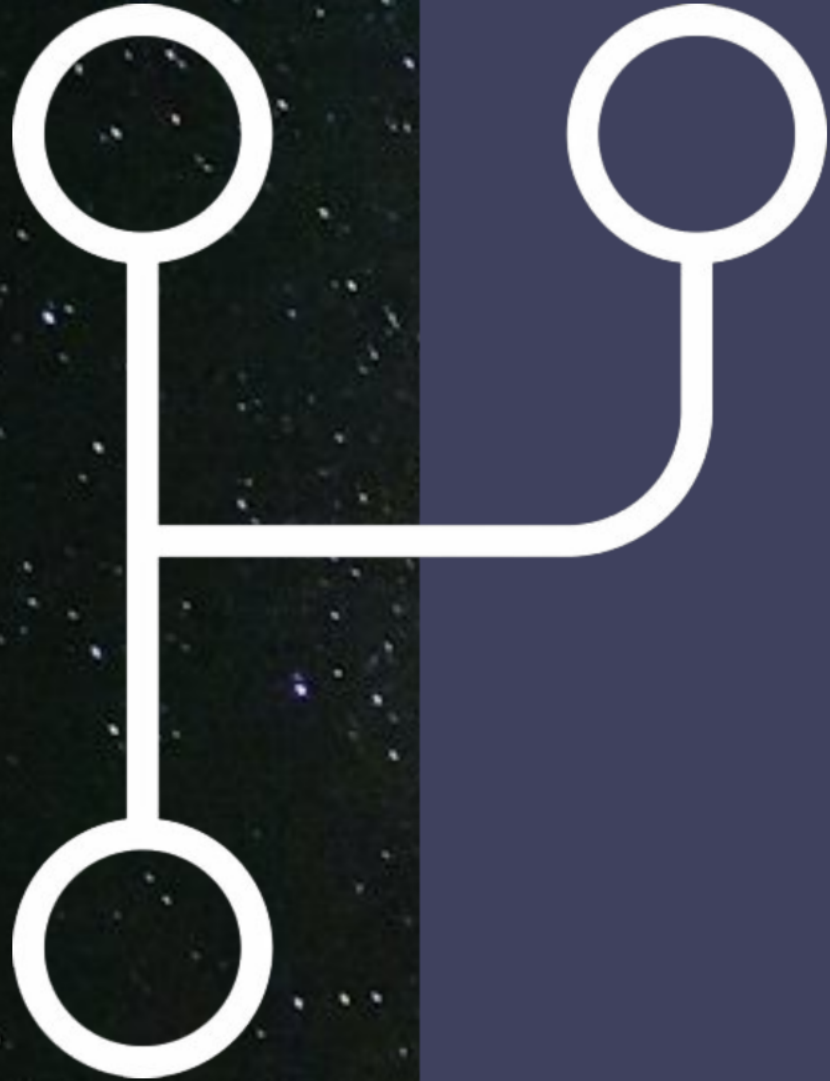
1. Regular updates on branch status, ongoing work, and potential conflicts.
2. Communication through platforms like GitHub can facilitate seamless collaboration.



Regular Merging Strategies

1. Isolate the different issues such as bugfix, feature, exploration, and documentation.
2. Implementing regular merging strategies.





Pulling Before Pushing

Always pulling the latest changes from the main branch before pushing your changes ensures your local repository is synchronized with the remote one.

Code Review Processes

1. A comprehensive code review process, checklist of reviewing with practice of Pull Request.
2. Allowing for feedback and insights from peers.

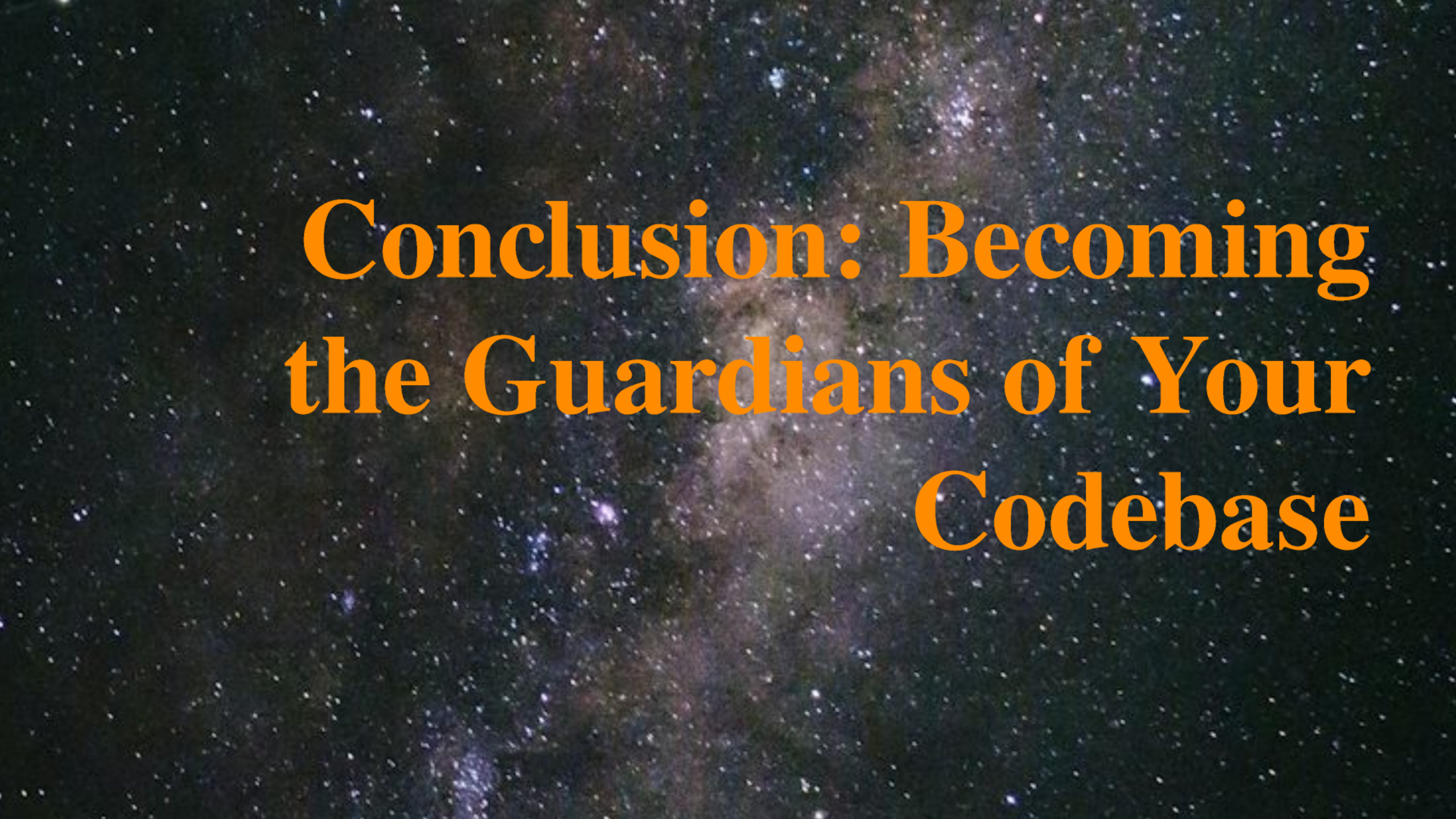


```
cout << "Enter row and columns for second matrix: ";  
cin >> r2 >> c2;  
  
// Taking elements of first matrix.  
cout << endl << "Enter elements of matrix 1:" << endl;  
for(i = 0; i < r1; ++i)  
{  
    for(j = 0; j < c1; ++j)  
    {  
        cout << "Enter element a" << i + 1 << j + 1 << " : ";  
        cin >> a[i][j];  
    }  
}  
  
// Taking elements of second matrix.  
cout << endl << "Enter elements of matrix 2:" << endl;  
for(i = 0; i < r2; ++i)  
{  
    for(j = 0; j < c2; ++j)  
    {  
        cout << "Enter element b" << i + 1 << j + 1 << " : ";  
        cin >> b[i][j];  
    }  
}
```


Integration and Continuous Development

uous
uous
actices
uality and
I/CD involves
merging code
branch, and
ne delivery.





Conclusion: Becoming the Guardians of Your Codebase

Summary of Git Management Principles

The main branch represents stability. With Git, don't worry about making mistakes.

Regular merging, pulling before pushing, and effective communication are all foundational principles essential for preventing conflicts and maintaining order.





The Role of Developers as Guardians

Developers play a critical role in protecting the codebase, akin to the TVA's role in the multiverse. By actively engaging in branch management practices, developers ensure that their work aligns seamlessly with the main branch, preventing the introduction of bugs or instability.

hts on egrity

a
he
rs. Just as
developers
to ensure
ne, thus
ready

```
Video = { type: "image" } || {}  
Url = { type: "url" } || {}  
Element = { type: "element" } || {}  
Object = { type: "object" } || {}  
  
check if boxer is already active, return  
"#boxer").length > 1 || !!(image  
return;  
  
l event  
event(e);  
  
the internal data  
$.extend({}, {  
window: $(window),  
body: $("body"),  
target: $target,  
object: $object,  
visible: false,  
resizeTimer: null,  
touchTimer: null,  
gallery: {  
  active: false  
}  
while || e.data.active
```




u ! Q&A Session

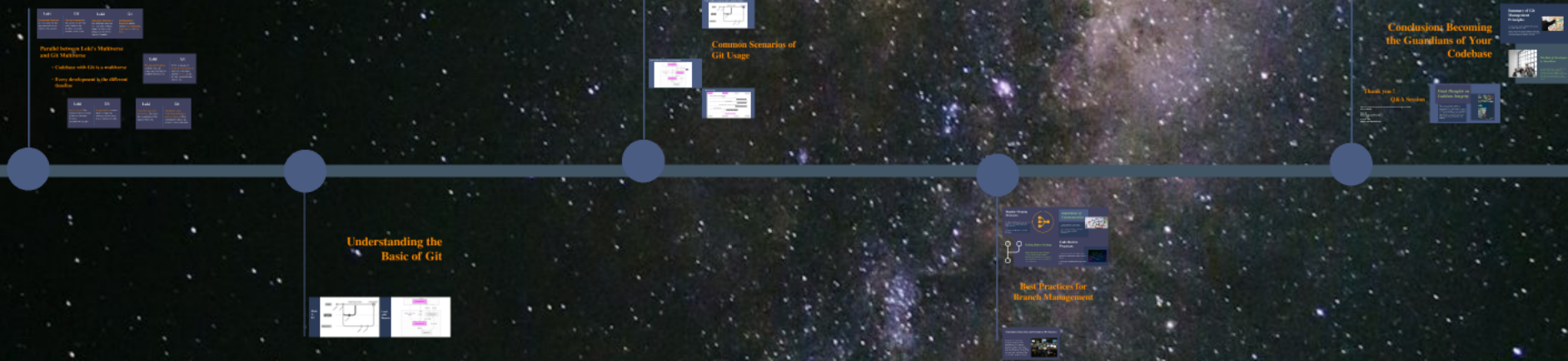
valued and encouraged. Contact

Final Thoughts Codebase Integrity

Maintaining codebase integrity is a continuous effort that requires the commitment of all team members. As the TVA prunes rogue timelines, we must regularly review our work for alignment with the Sacred Timeline and constructing a stable production-ready codebase.

Be the Cosmic Guardian of Your Codebase with Git

Navigating Git Branches with



PHUSE EU Connect 2024 – OS15

Author: Shuai Wu, Brian M. Lang, Carl Herremans,
Christian Koul, MSD