Abstract

Fabric, Git, & Pipelines - A Chord in Harmony

Nowadays with the rise of Microsoft Fabric a data engineer has the option to store data in warehouses, lakehouses, or KQL databases. As if we had to tell you that, that's the natural order of things. But there are other things a data engineer needs that do not belong in that kind of storage. Where do the SQL scripts, the notebooks, or the KQL query sets go?

If you don't know better, your local hard drive. If you are very bold, you might store them in Microsoft SharePoint. Or what's good enough for your data is good enough for your scripts, after all this saves on storage costs. Luckily, there is a better option: Git.

The scripts and notebooks are stored in a Git repository. Git is a distributed version control system initially developed for the Linux kernel. It allows for easily versioning your files and for collaboration on the same notebook. Git is a tool that has many amazing features, but like any tool it requires knowledge to use. Make your first steps into Git as a data engineer here and learn what Git can do for you when you develop a data solution. We show working with Git inside and outside of Microsoft Fabric with a focus on applicability and best practices.

Ok, now that your SQL scripts and your Fabric notebooks are fully versioned with Git, how to get them from the Git repository onto the Fabric workspaces? Of course, you could copy them manually, but wouldn't it be nicer if they just deployed automagically to the right place whenever a change occurs? The DevOps world has a solution: DevOps Pipelines. Learn how a pipeline can take your notebook and other Fabric items and deploy them to dev, staging, and prod environments, adapting connection strings and other parameters to match each environment automatically. We demonstrate with Microsoft Fabric how to automate your workflow so you can directly see the benefit.

Git and DevOps pipelines have helped software engineering immensely. And it might just do the same for data. Taking the load off your back by automating tasks in Microsoft Fabric can make your daily life easier. And working properly with version control gives you safety and recovery from error for your Fabric items. Let Git and Pipelines shine together to make your Fabric endeavor brighter

Fabric, Git, & Pipelines - A Chord in Harmony

Who are we?

Marisol Steinau

- Autodidact Data Enthusiast
- > 8 years experience using Microsoft Data Platform
- Data Solution Architect
- Frequent guest at Legoland Resort Germany
- linkedin.com/in/marisol-steinau-bb1253253
- marisol.steinau@steinautech.com

Sebastian Steinau

- Techie (Azure, .NET, DevOps, IaC)
- Stepping into the data world with Fabric
- Cloud Solution Architect
- TV-Series geek and cook
- linkedin.com/in/sebastian-steinau-312888200
- sebastian.steinau@steinautech.com

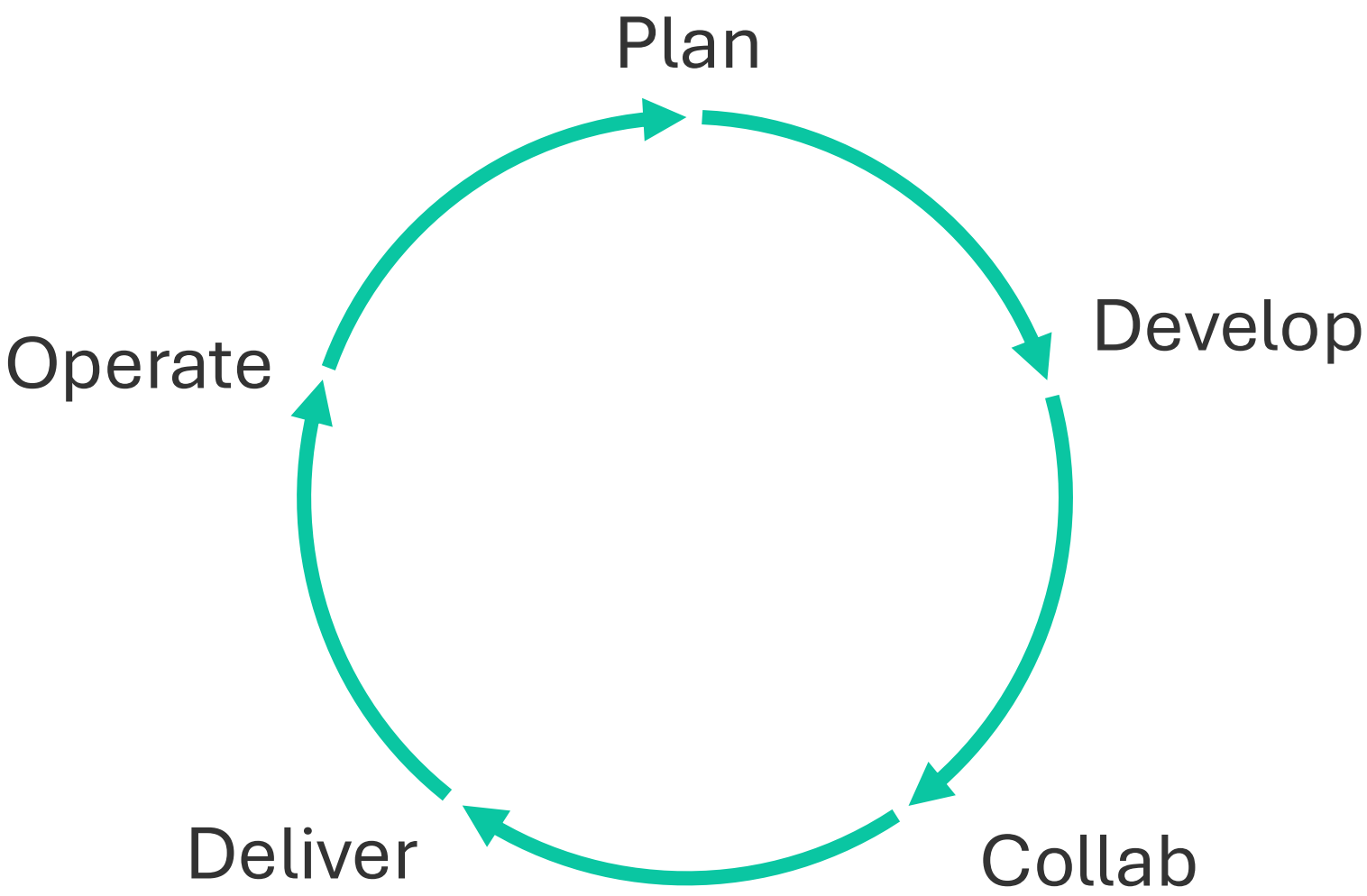
Joint blog: refugeinaudacity.eu

(Work in progress)

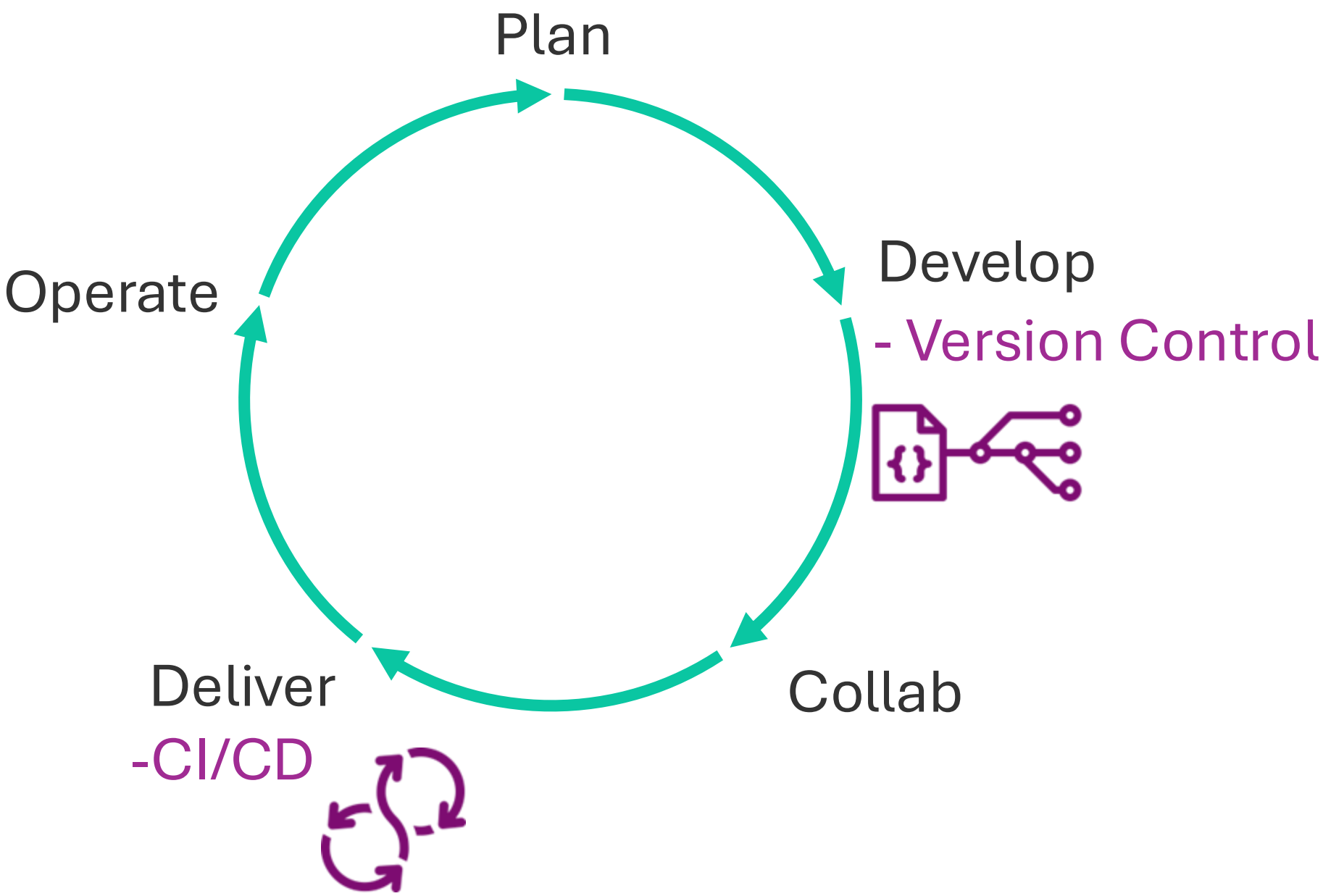
Agenda

- Introduction to DevOps
- Version control with Git
- Git in Fabric
- CICD in Fabric
- Improving Fabric with DevOps Pipelines
- Conclusions

DevOps



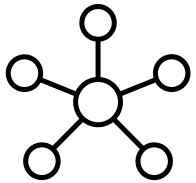
DevOps



Advantages of Version Control

Versionable

If something doesn’t work, you can go back to a working version



Unambiguous

A central place acts as a single point of truth

Traceability

What changes were made, who made them and why allows to track progress and resolve issues

Integrity

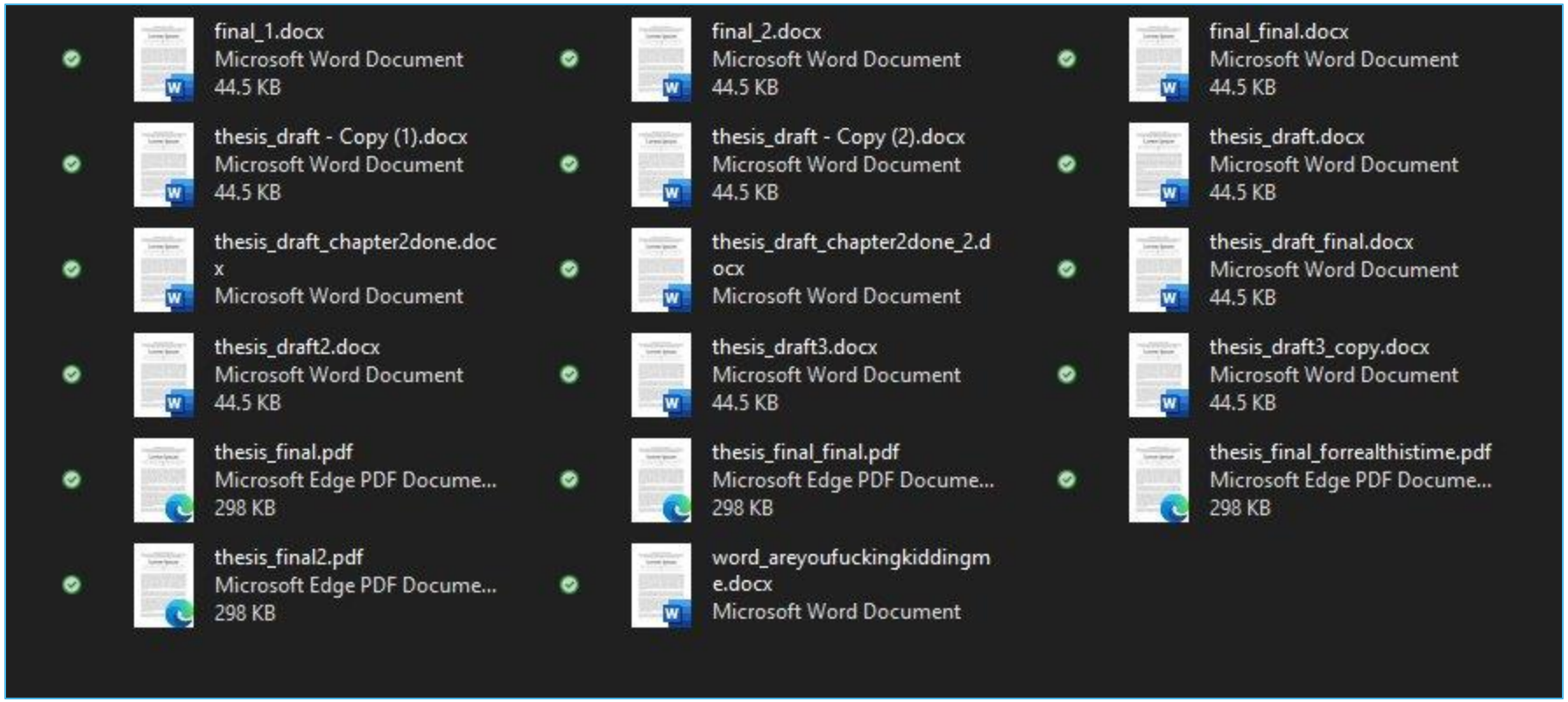
The code or files stored in the system remain unaltered and uncorrupted

Version Control

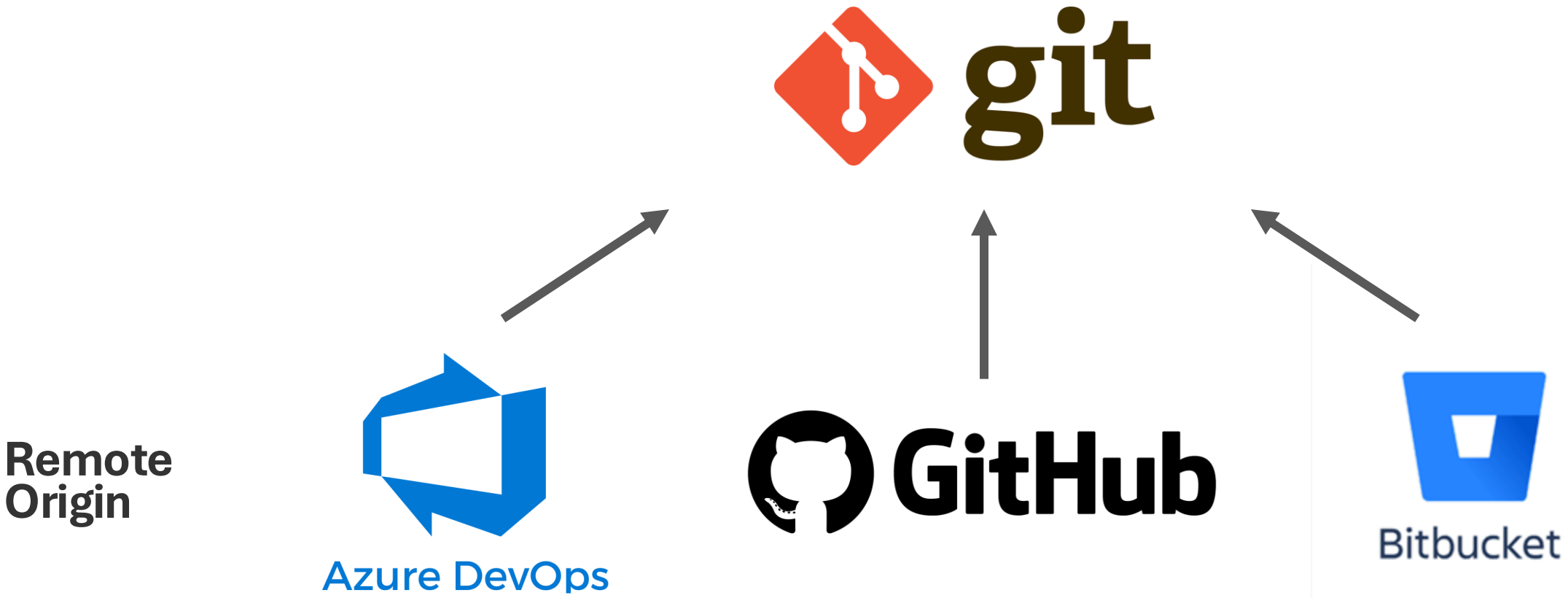
Do I need version control If I am a team of one?

Version Control

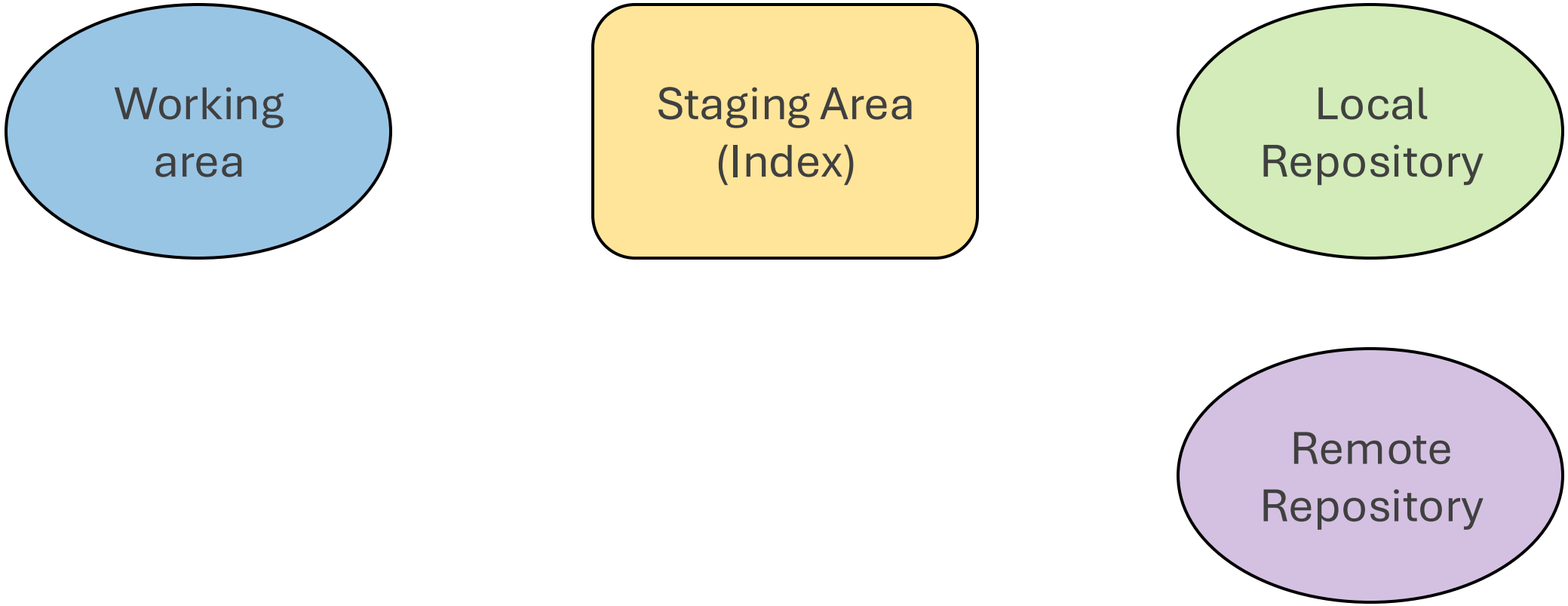
Do I need version control If I am a team of one?



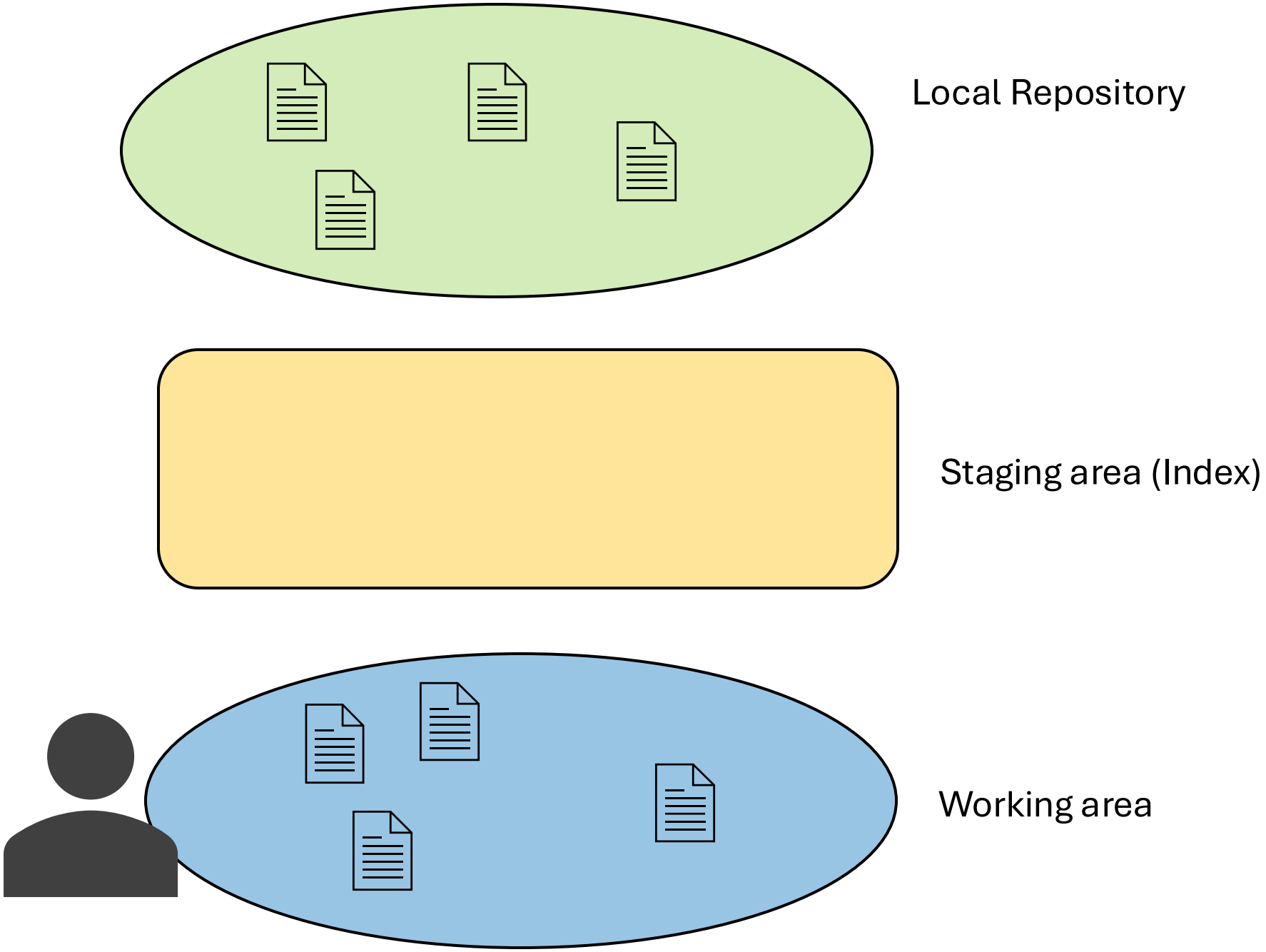

git
Versioning
Enables collaboration
Source of Truth
Consistency
Immutable history


git
Remote
Origin
Azure DevOps
GitHub
Bitbucket

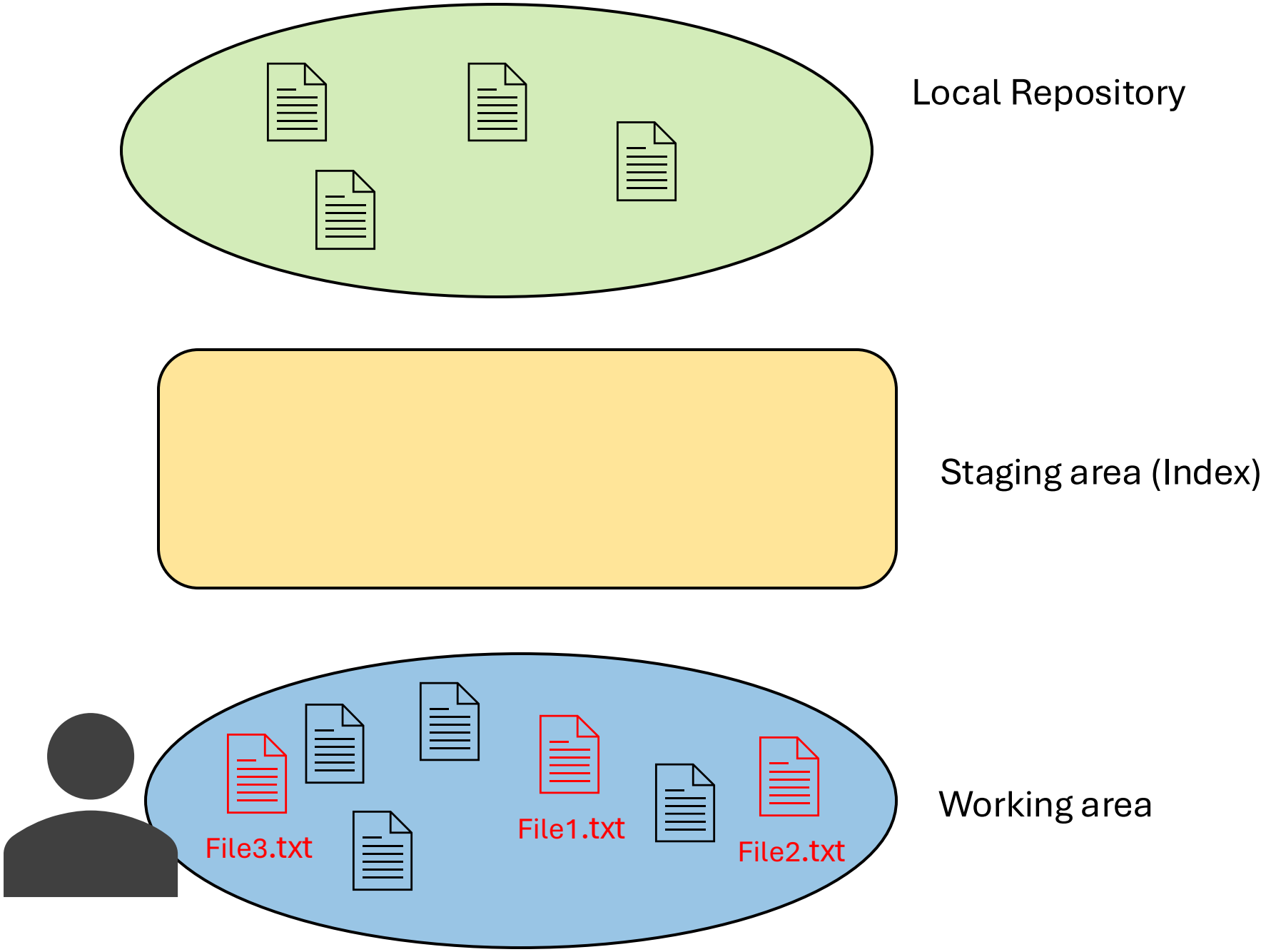
Git


Working area
Staging Area (Index)
Local Repository
Remote Repository

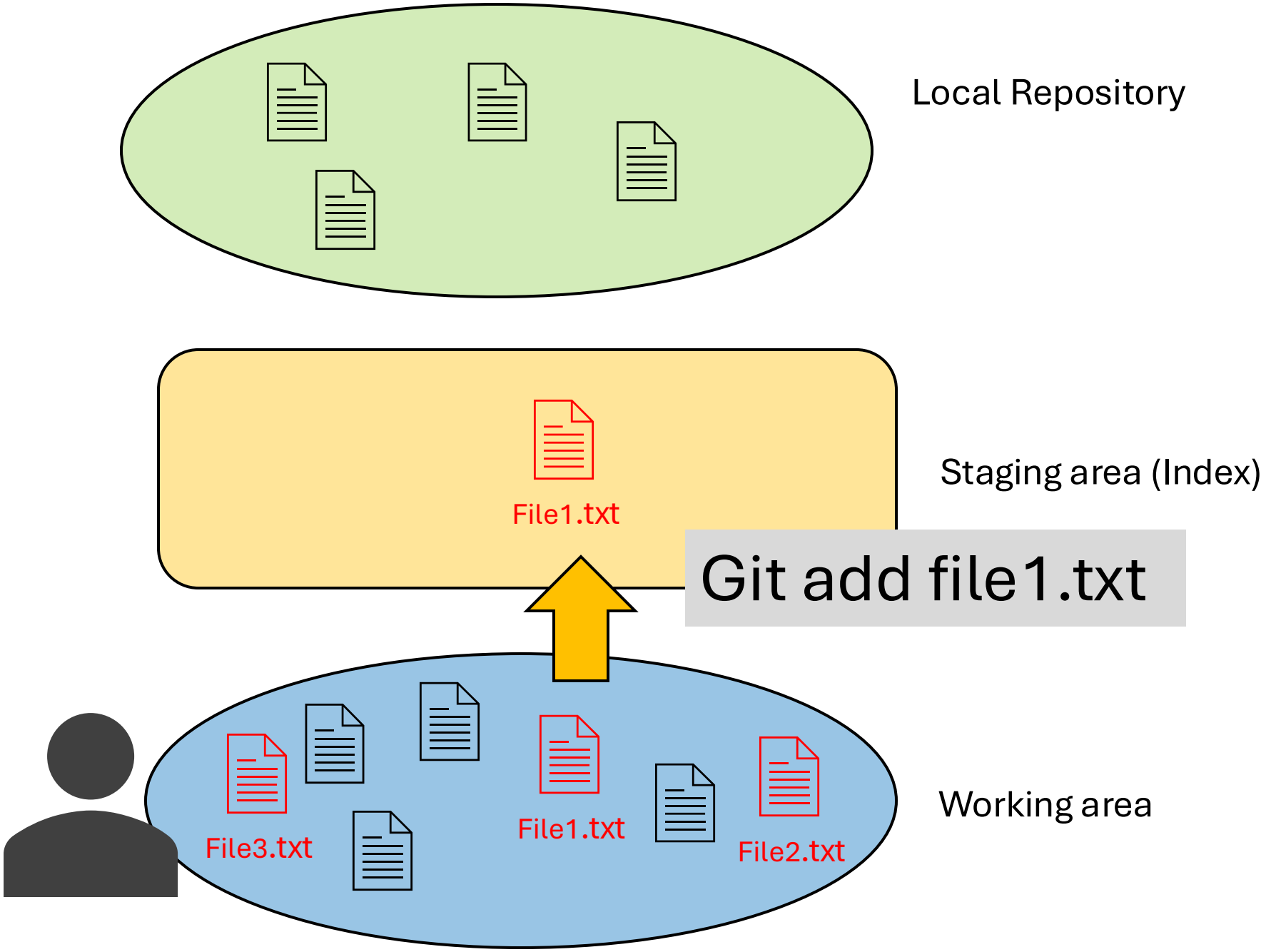
Git



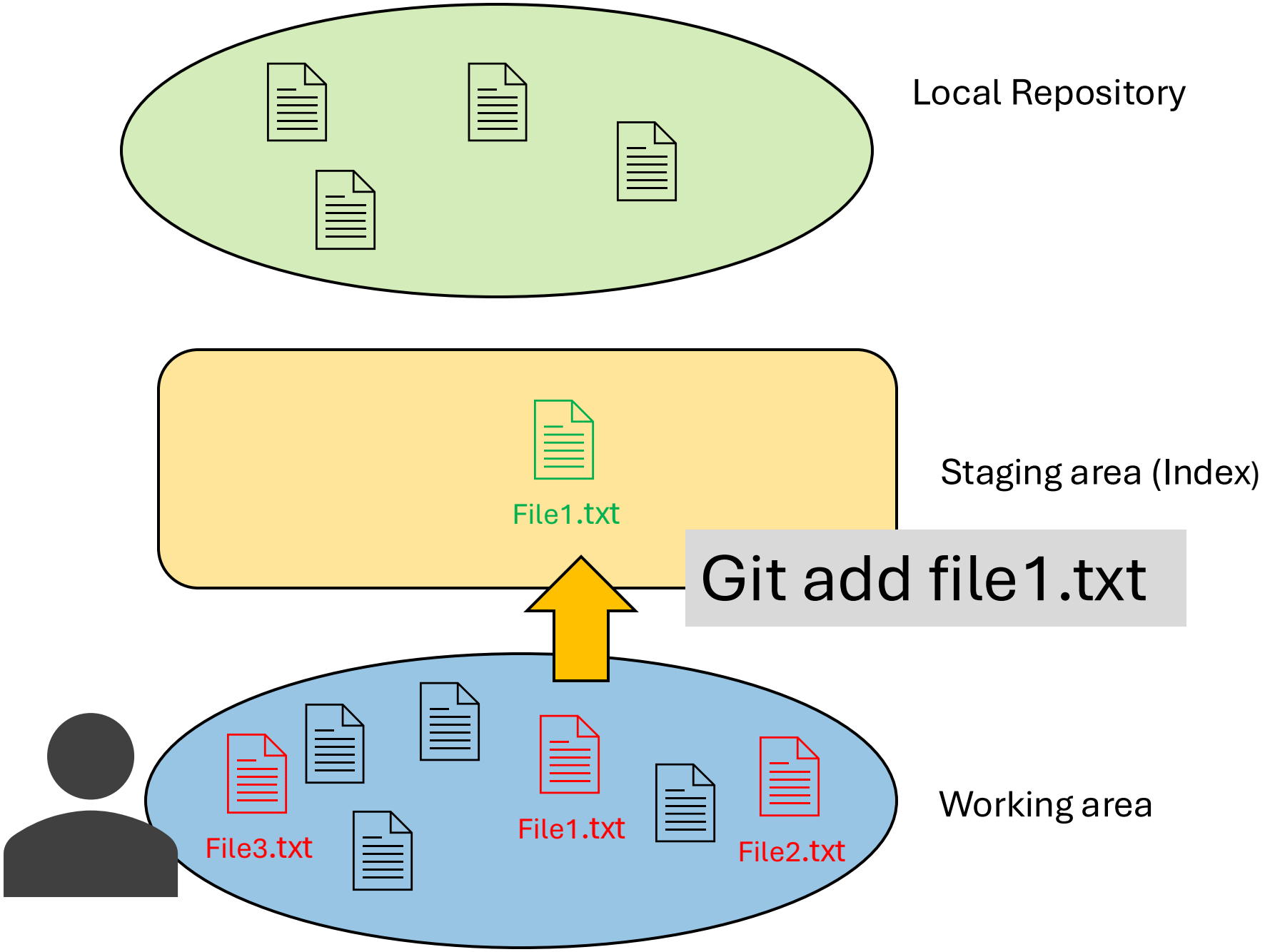
Git


Local Repository
Staging area (Index)
Working area
File3.txt
File1.txt
File2.txt

Git


Local Repository
File1.txt
Staging area (Index)
Git add file1.txt
File3.txt
File1.txt
File2.txt
Working area

Git


Local Repository
File1.txt
Staging area (Index)
Git add file1.txt
File3.txt
File1.txt
File2.txt
Working area

Git

Local Repository

Git commit –m „Added new file“

File1.txt

Staging area (Index)

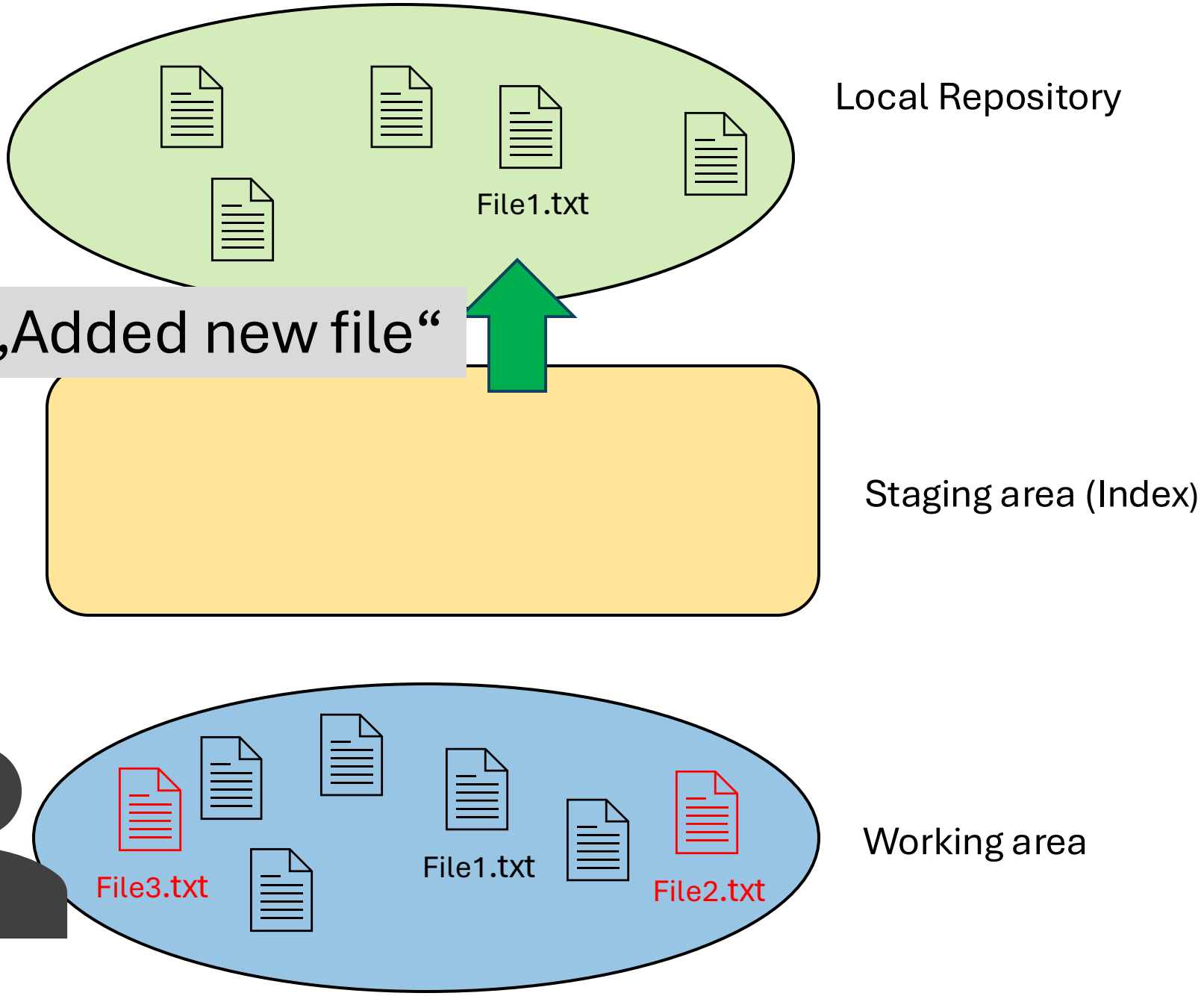
File3.txt

File1.txt

File2.txt

Working area

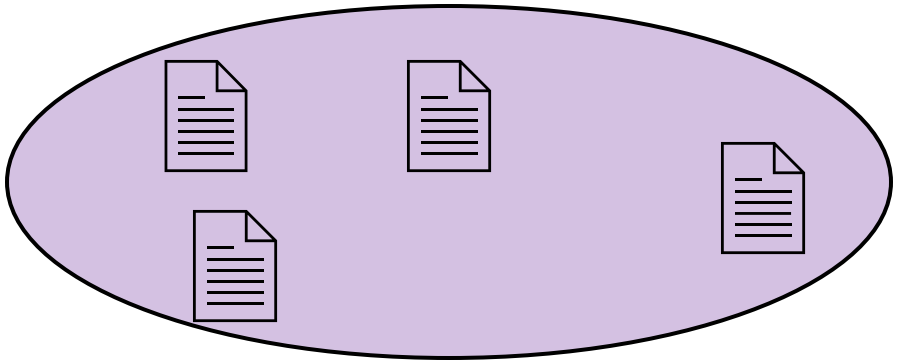
Git

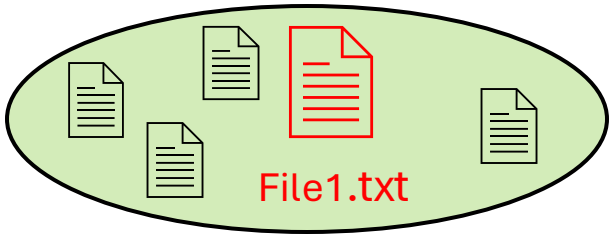


Git commit –m „Added new file“

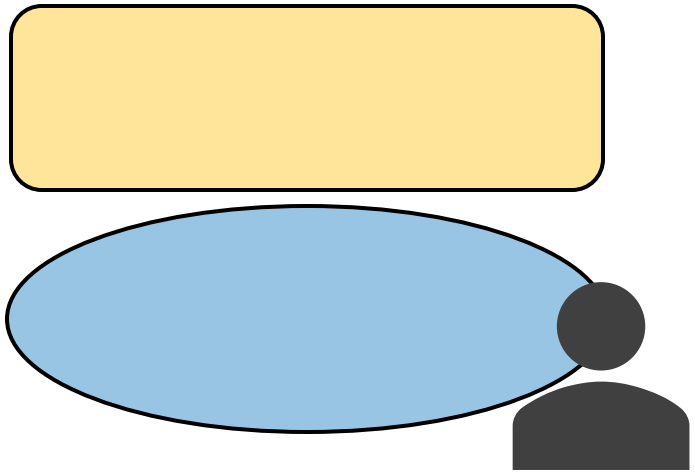
Git

Remote Repository

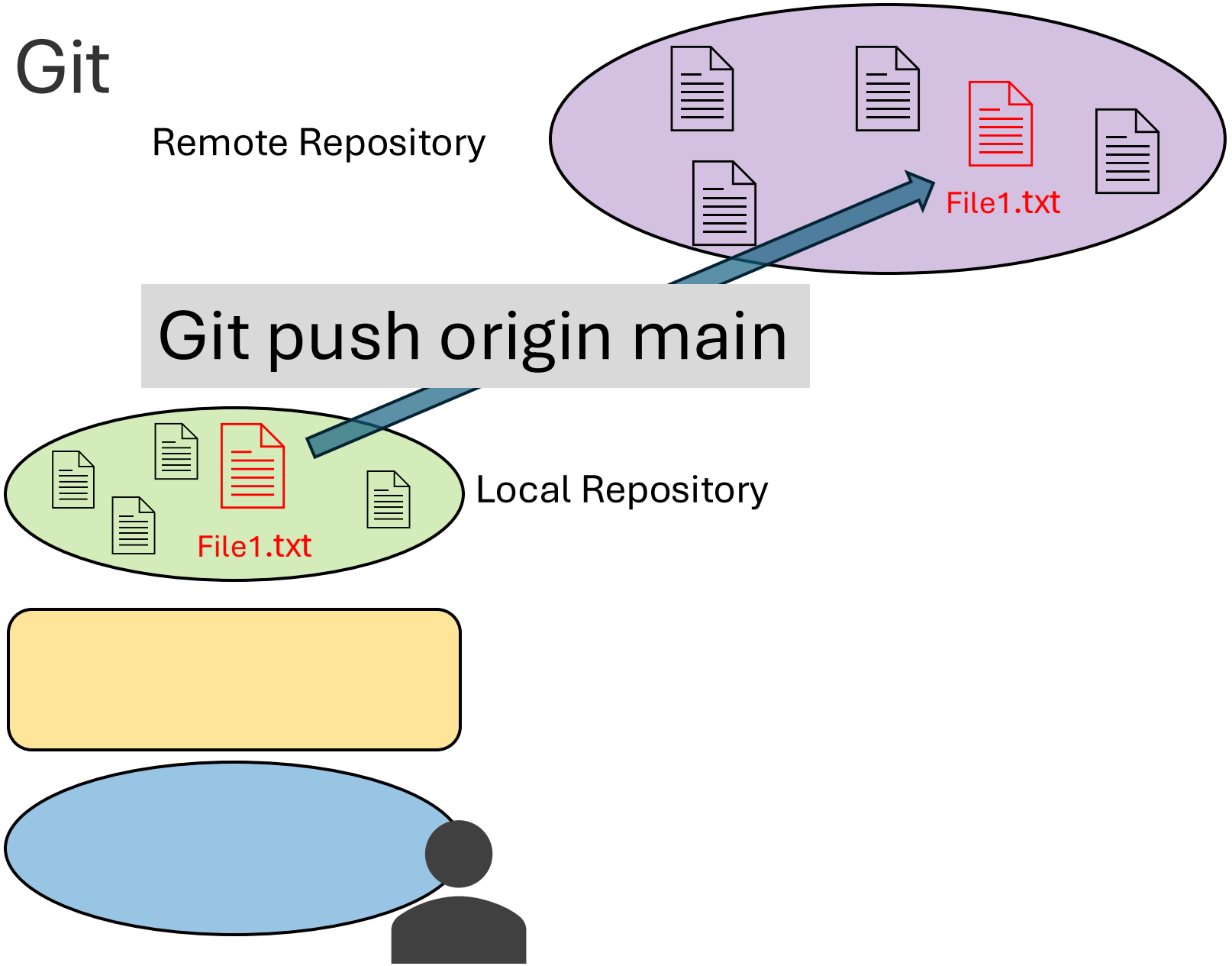




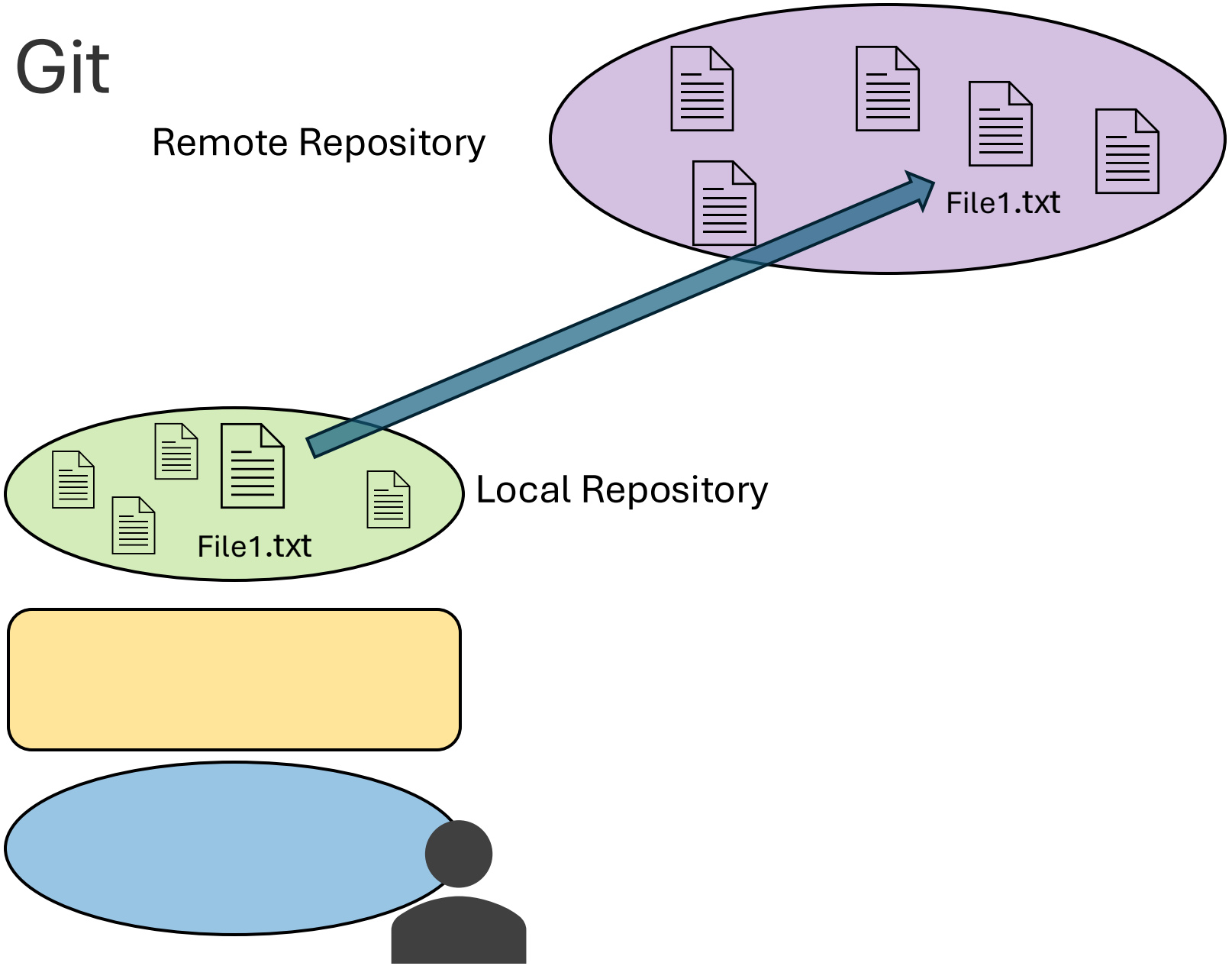
Local Repository



Git

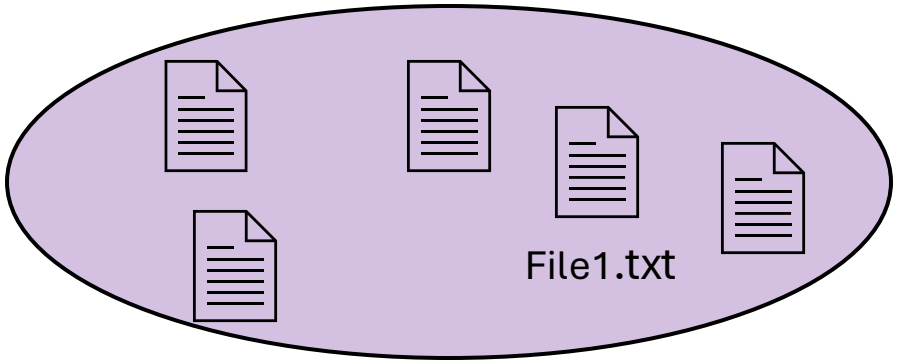

Remote Repository
File1.txt
Git push origin main
File1.txt
Local Repository

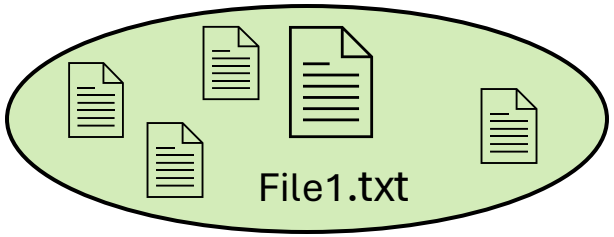
Git



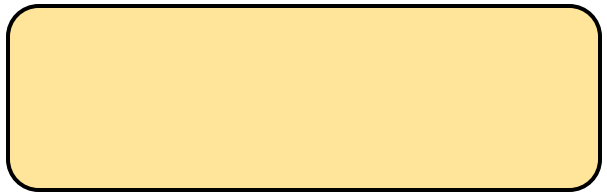
Git

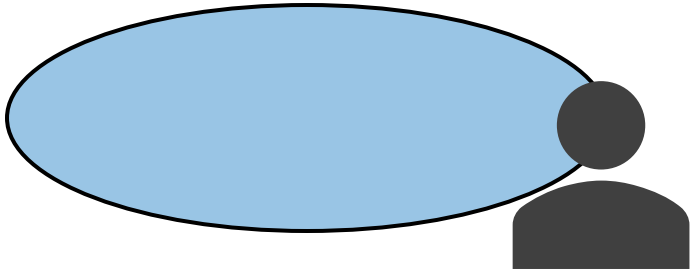
Remote Repository





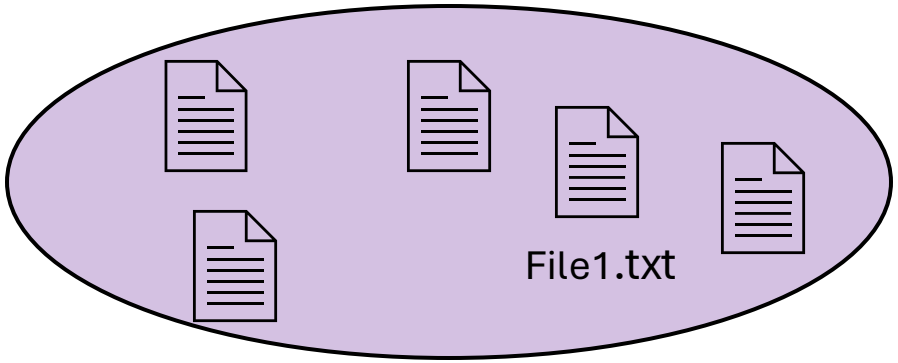
Local Repository

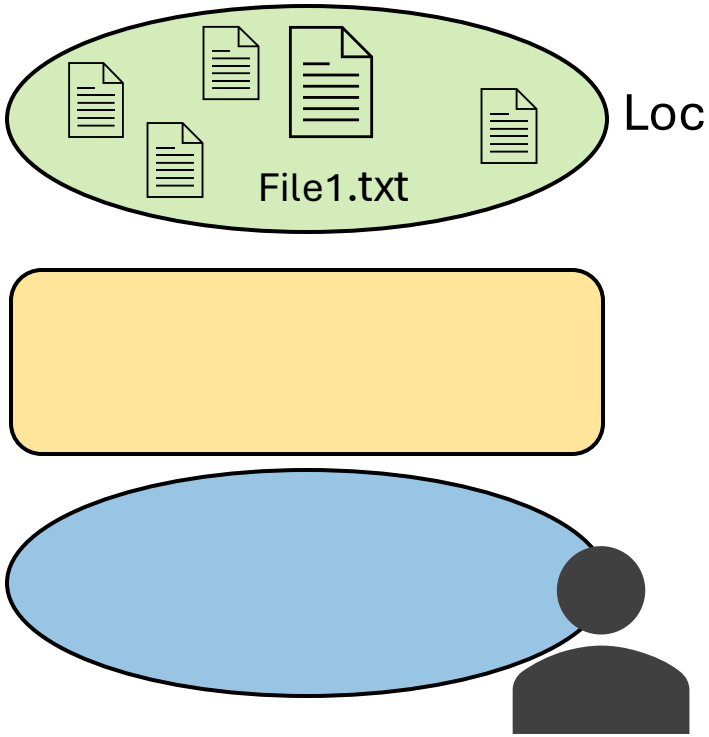




Git

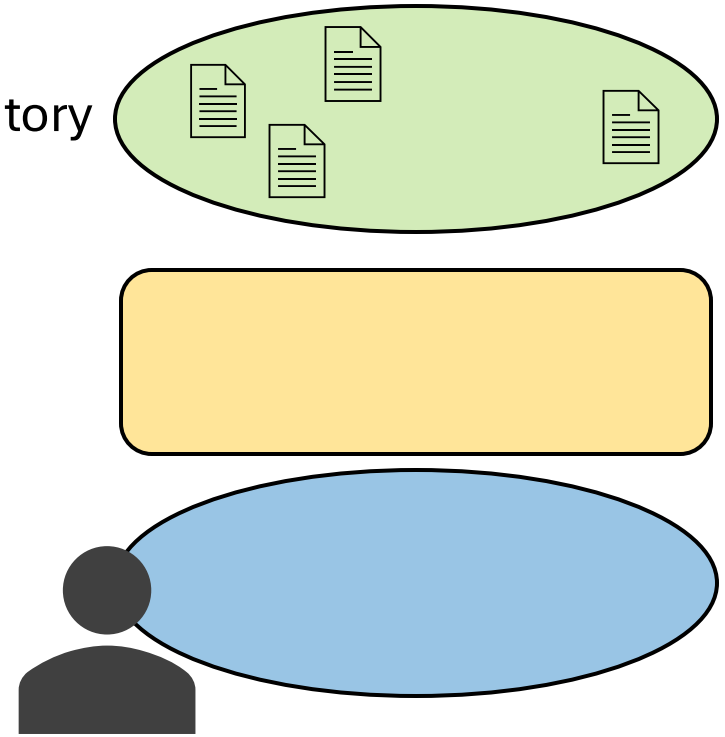
Remote Repository



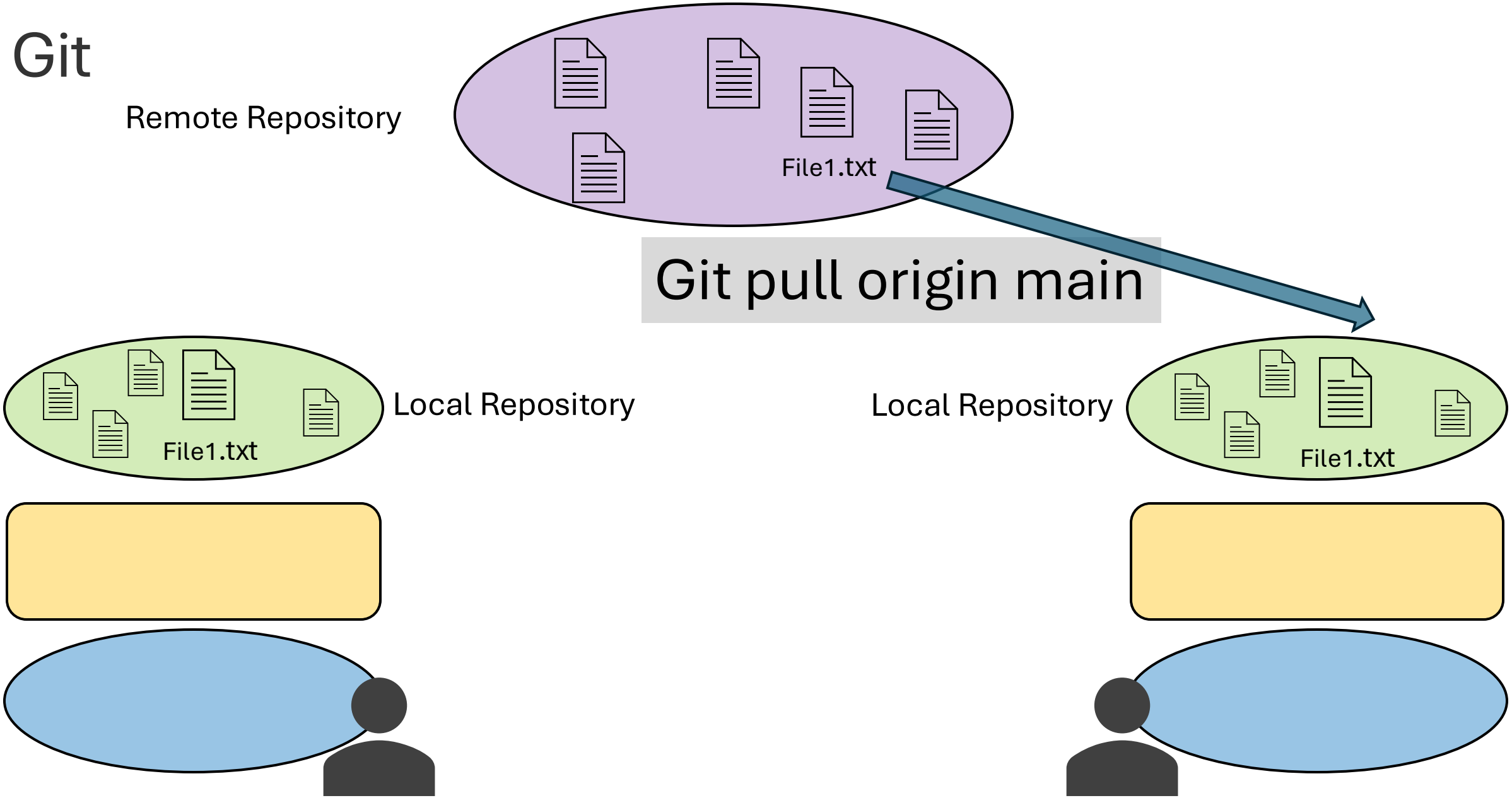


Local Repository

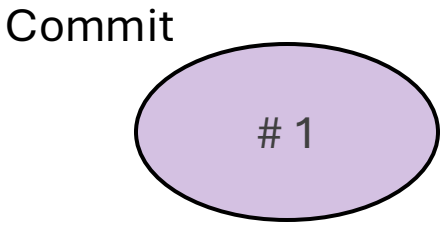
Local Repository



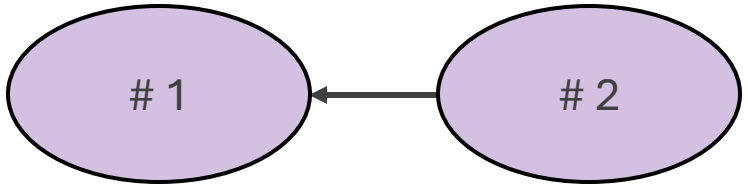
Git


Remote Repository
File1.txt
Git pull origin main
Local Repository
File1.txt
Local Repository
File1.txt

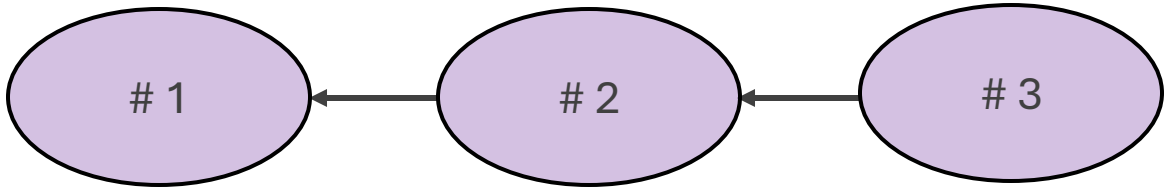
Branches



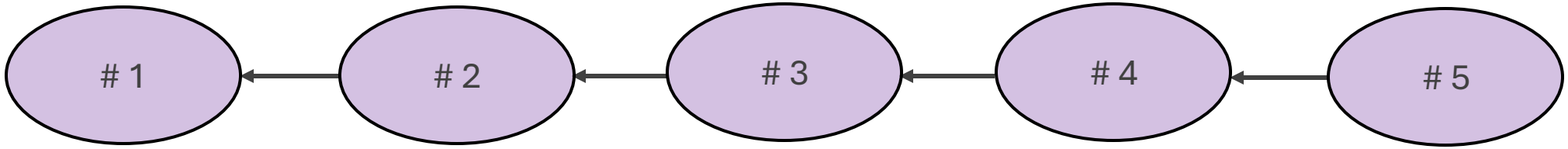
Branches



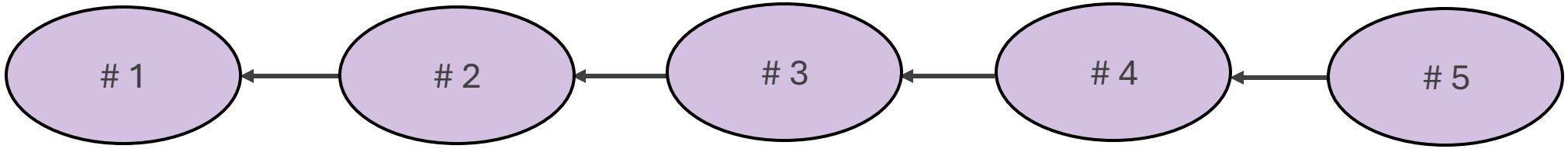
Branches



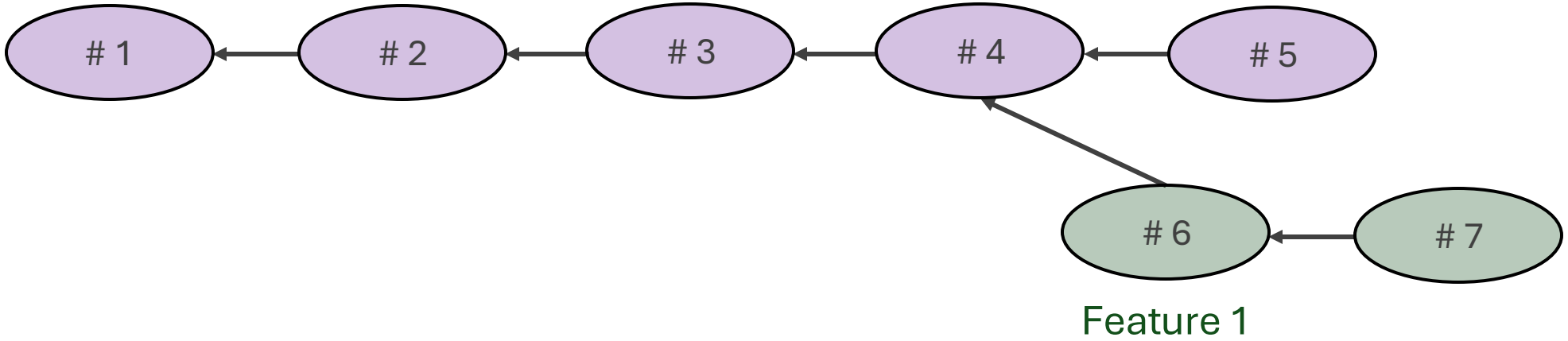
Branches



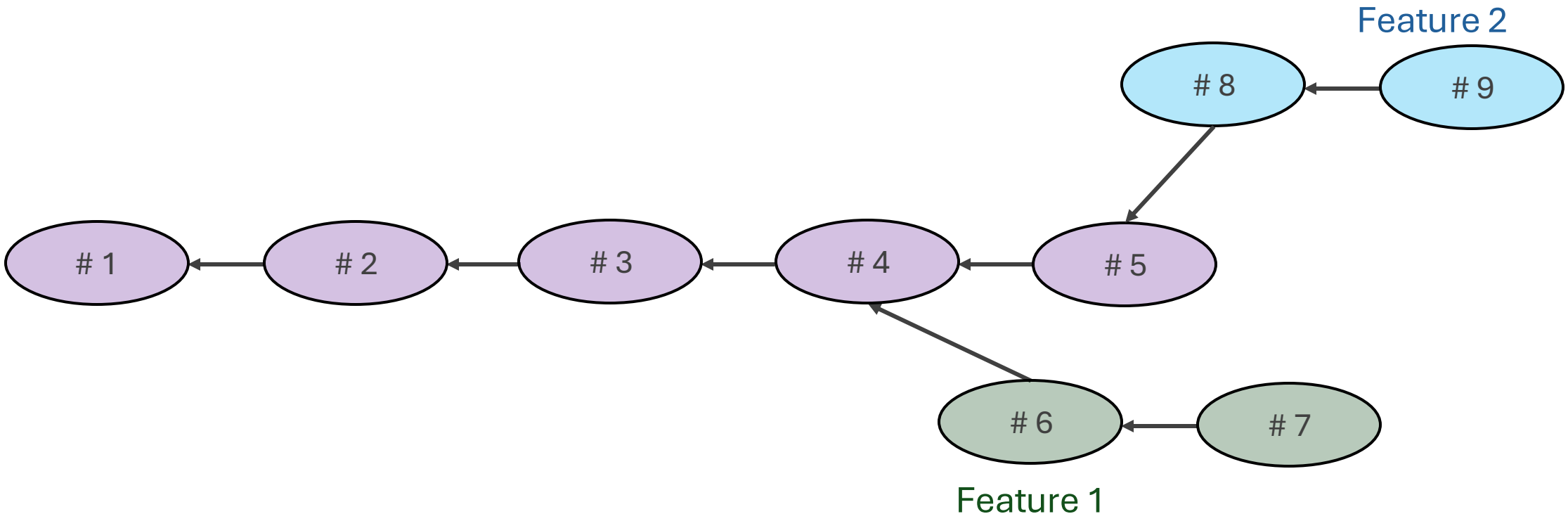
Branches



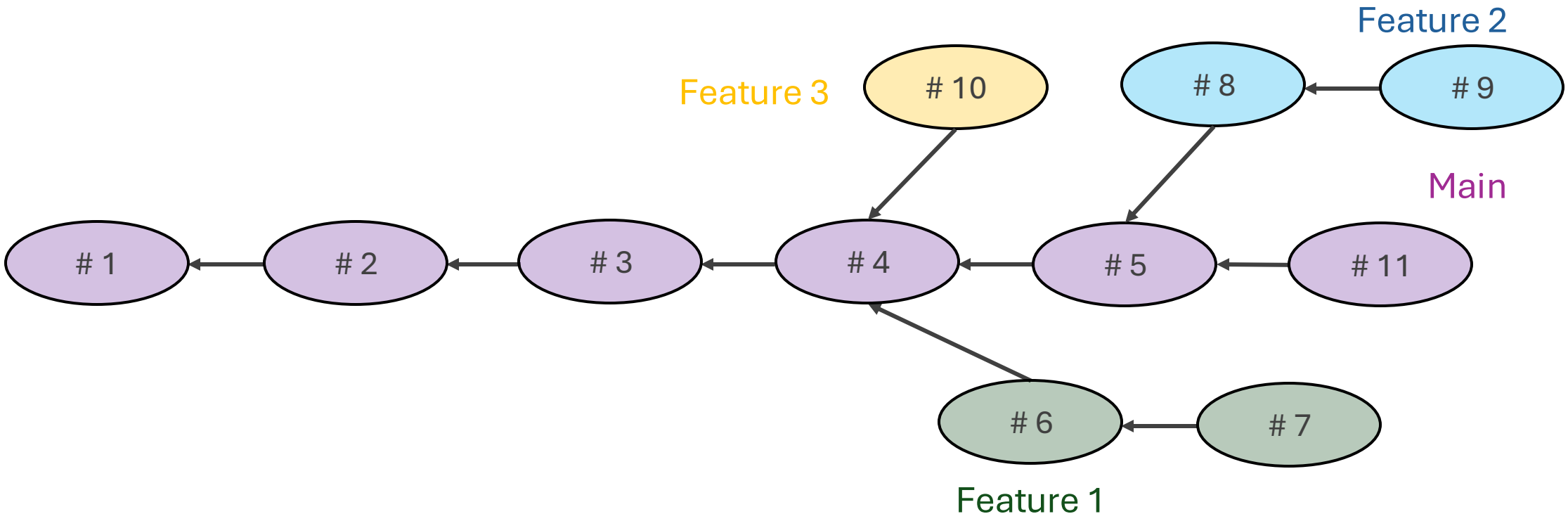
Branches



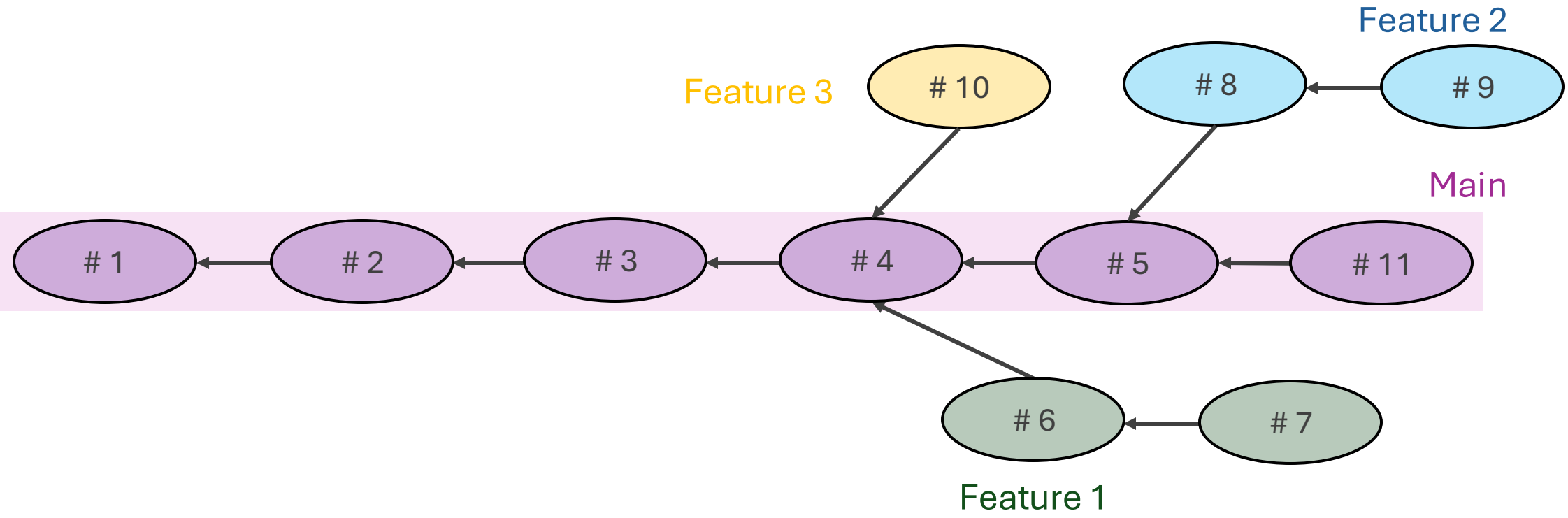
Branches



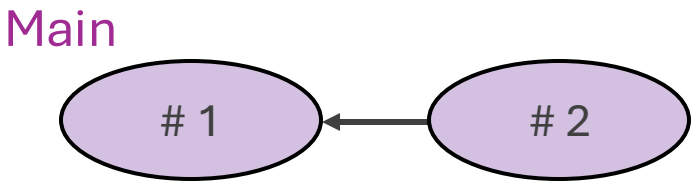
Branches



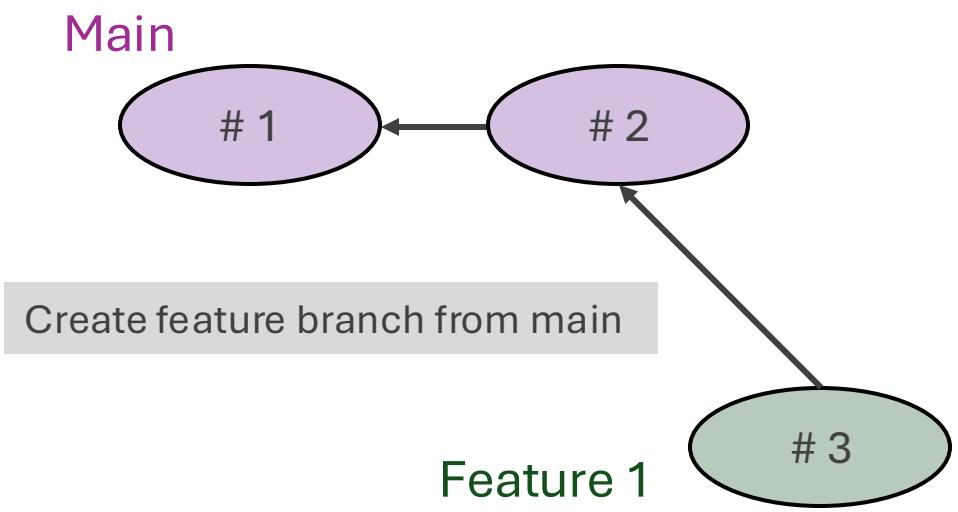
Branches



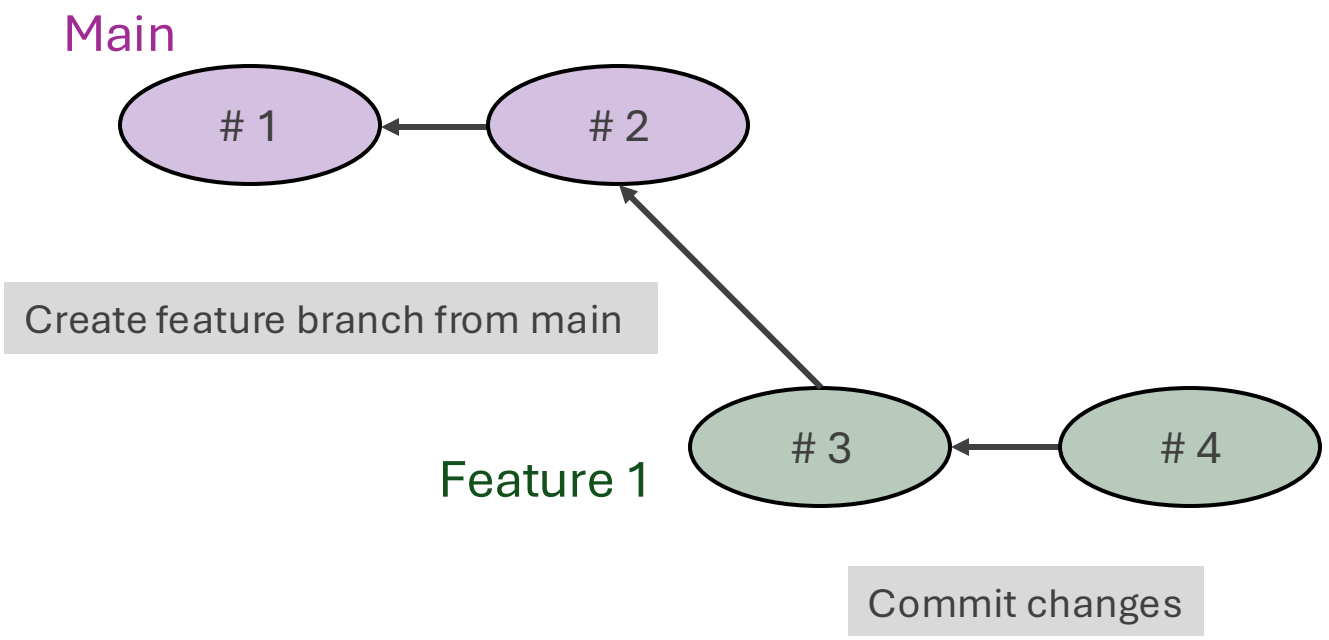
Branches



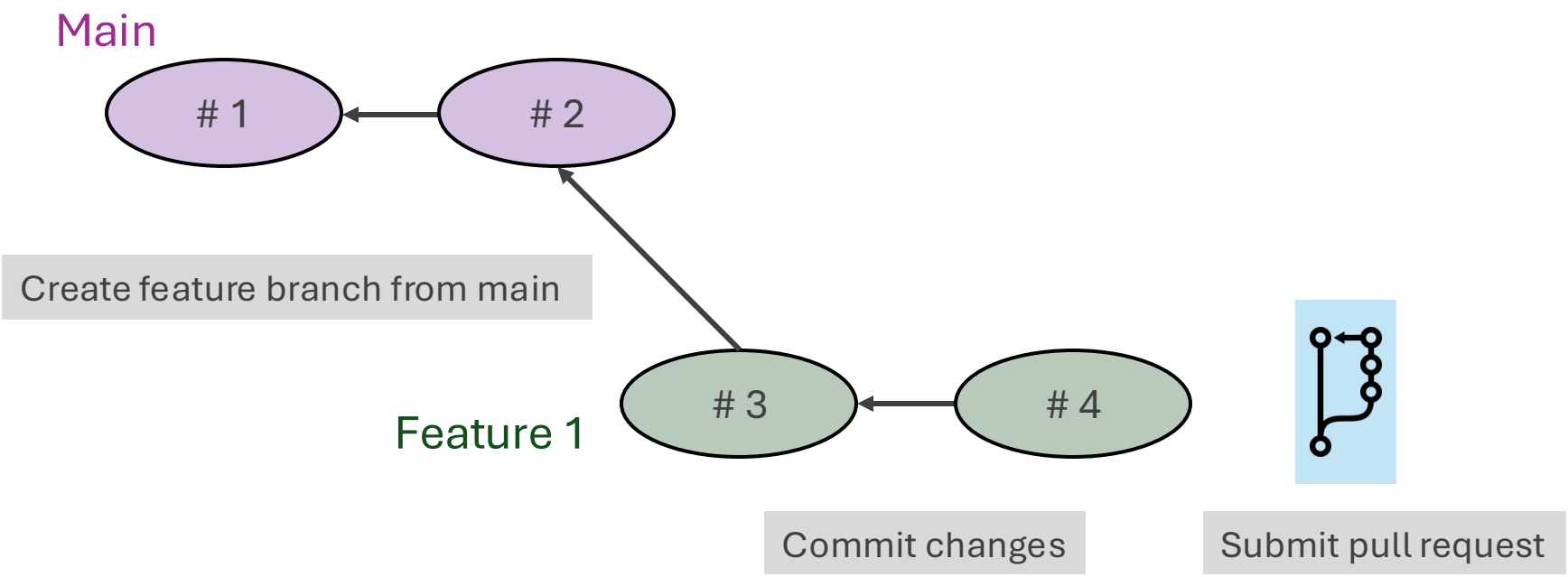
Branches



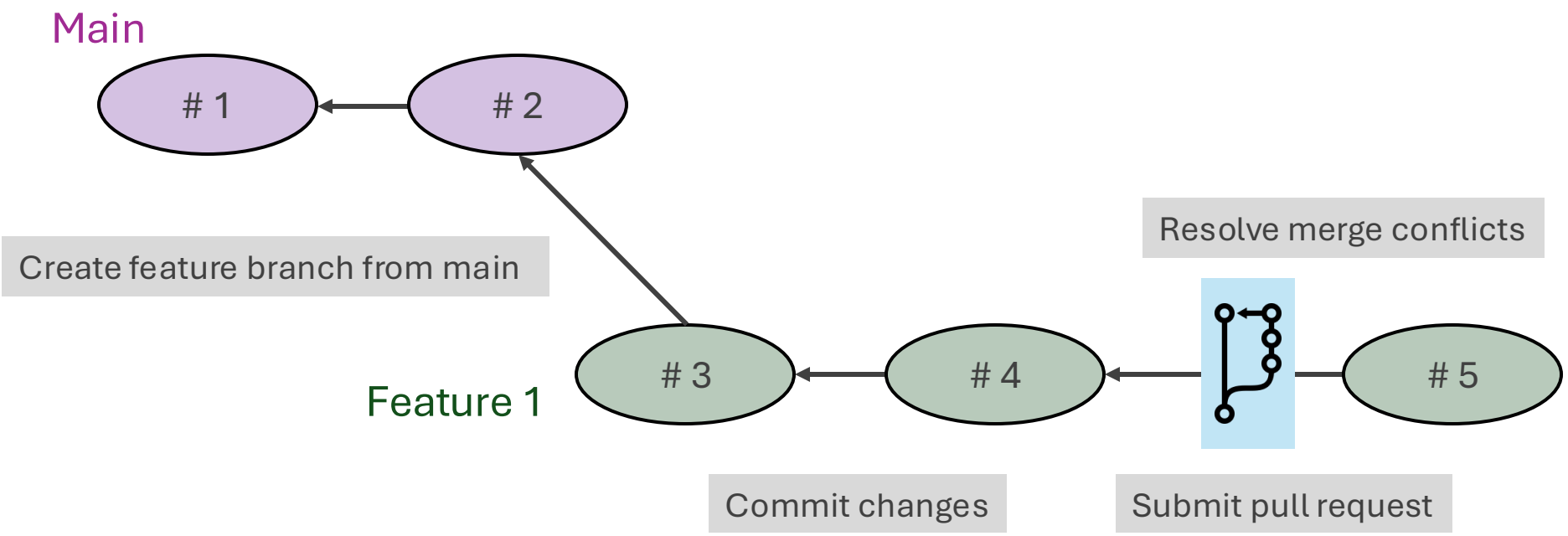
Branches


Main
1
2
Create feature branch from main
Feature 1
3
4
Commit changes

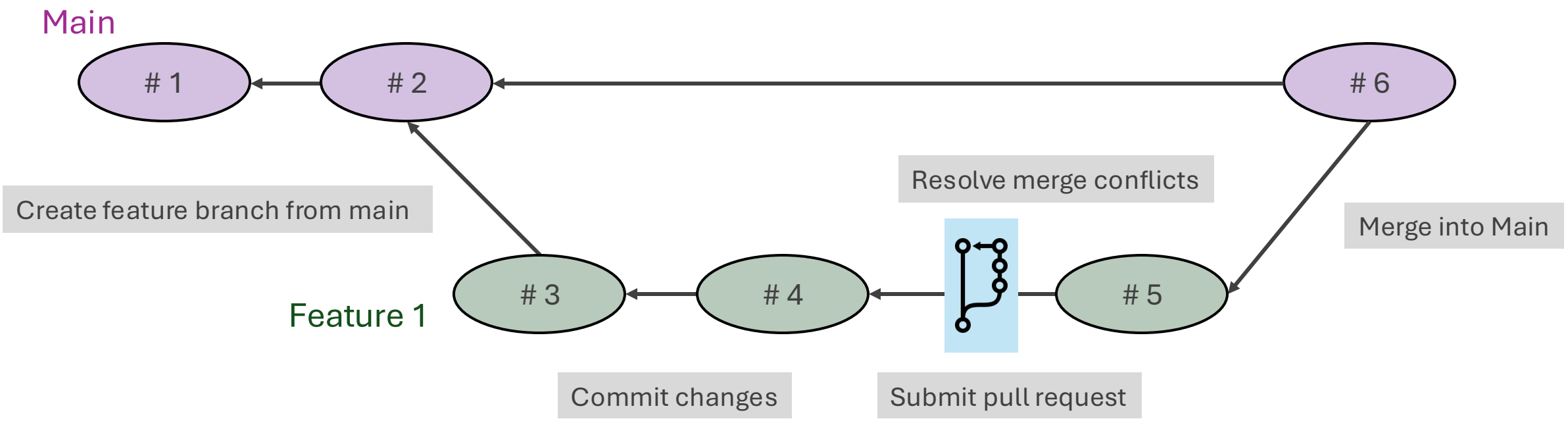
Branches


Main
1
2
Create feature branch from main
Feature 1
3
4
Commit changes
Submit pull request

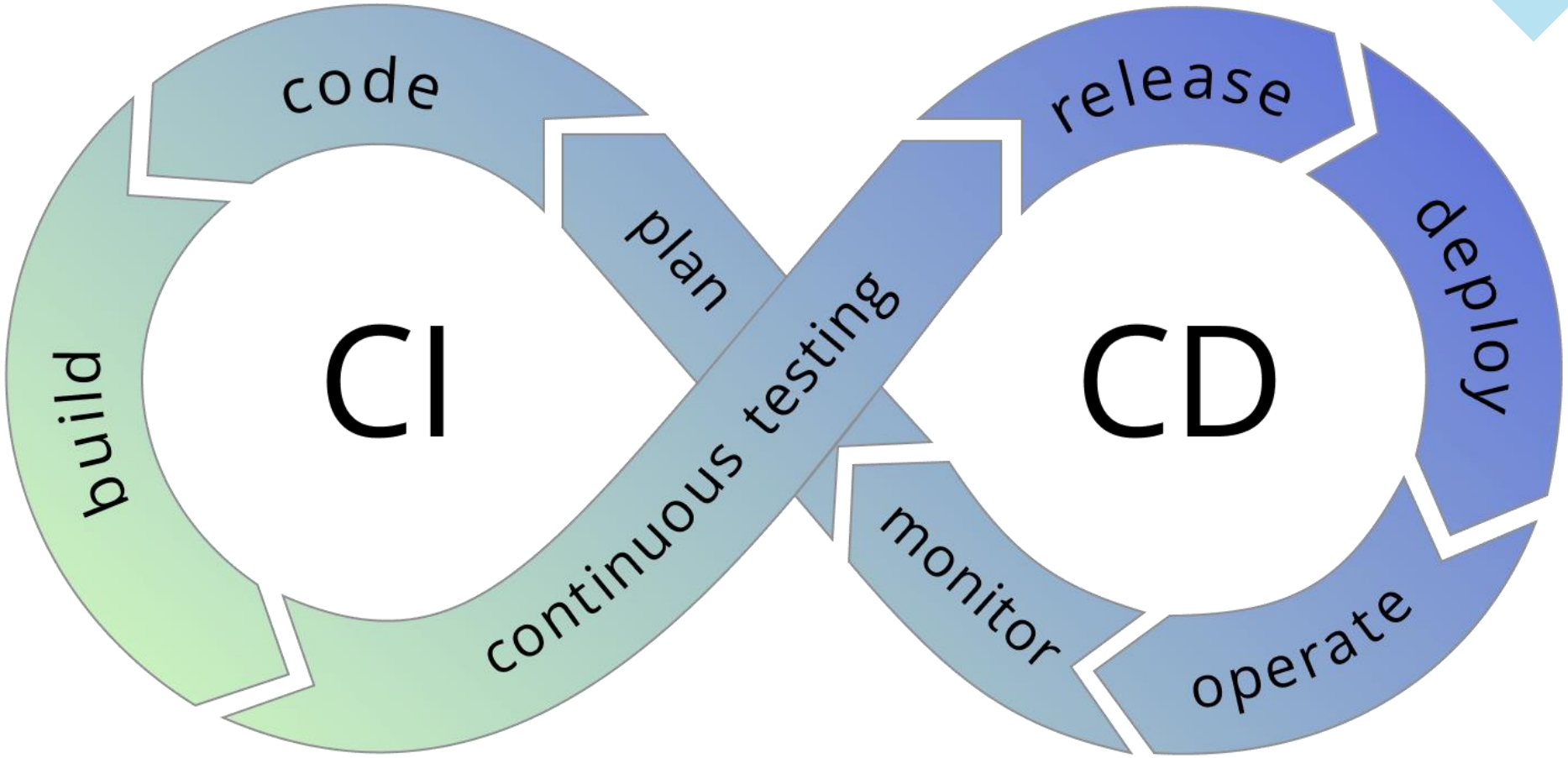
Branches


Main
1
2
Create feature branch from main
Resolve merge conflicts
Feature 1
3
4
5
Commit changes
Submit pull request

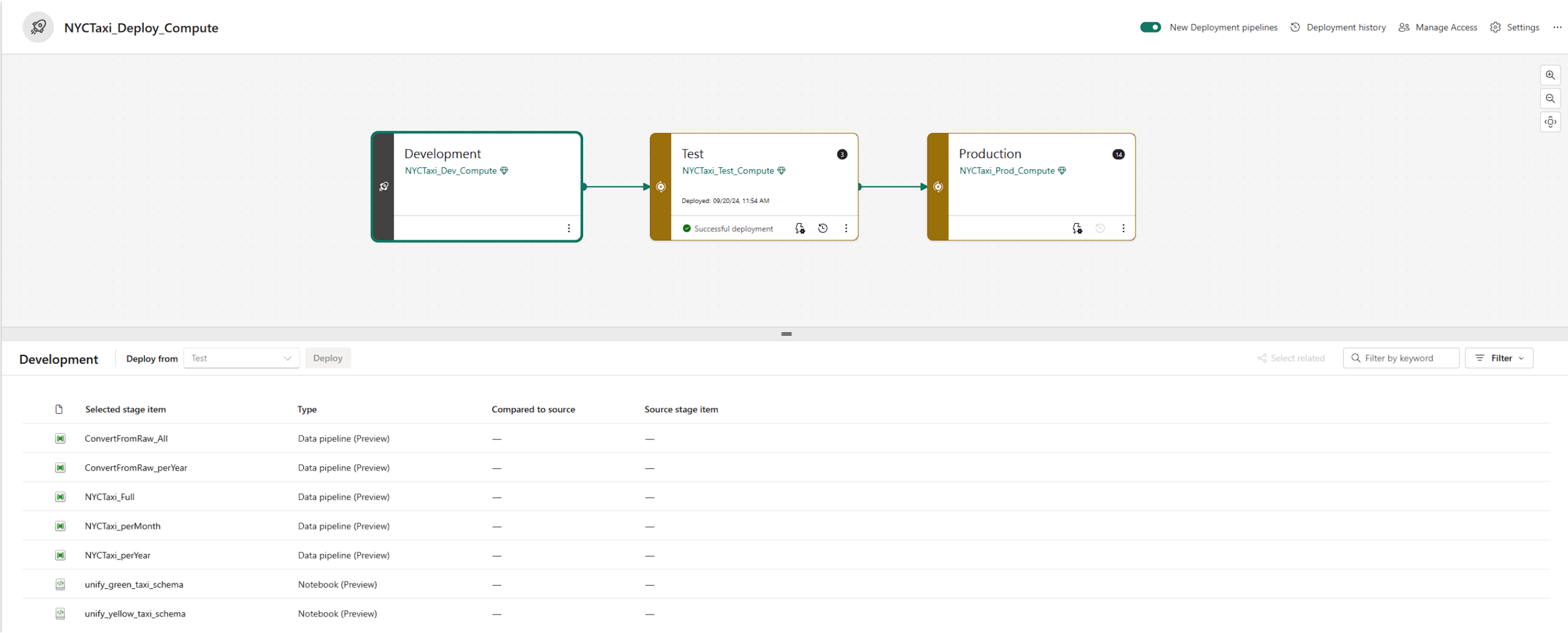
Branches


Main
1
2
6
Create feature branch from main
Resolve merge conflicts
Merge into Main
Feature 1
3
4
5
Commit changes
Submit pull request

Git in Action


code
plan
release
deploy
CI
CD
build
continuous testing
monitor
operate

Deployment Pipelines



CI/CD in Microsoft Fabric

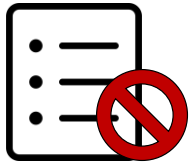
Drawbacks

No Git synchronization enforcement

Workspaces are not automatically synced with Git before or after deployment

Triggered Manually

Prone to human errors



Limited support for deployment rules

Certain configurations are not parametrizable during deployment, forcing to manually adjust some items post-deployment

CI/CD in Microsoft Fabric

Limitations

Fabric Item	Git Integration	Deployment Pipelines
Lakehouses	x (partially)	x
Warehouses	x (bugged)	x (bugged)
Data Pipelines	x	x
Notebooks	x	x
Paginated Reports	x	x
Spark Job Definitions	x	
Spark Environments	x	x
Reports	x	x
Semantic Models	x	x
Dataflows Gen2		
Datamarts		
Dashboards		
Eventhouses		Announced
Eventstreams		
KQL Database		
KQL Queryset		
ML Model		
ML Experiment		

CI/CD in Microsoft Fabric

Desired features

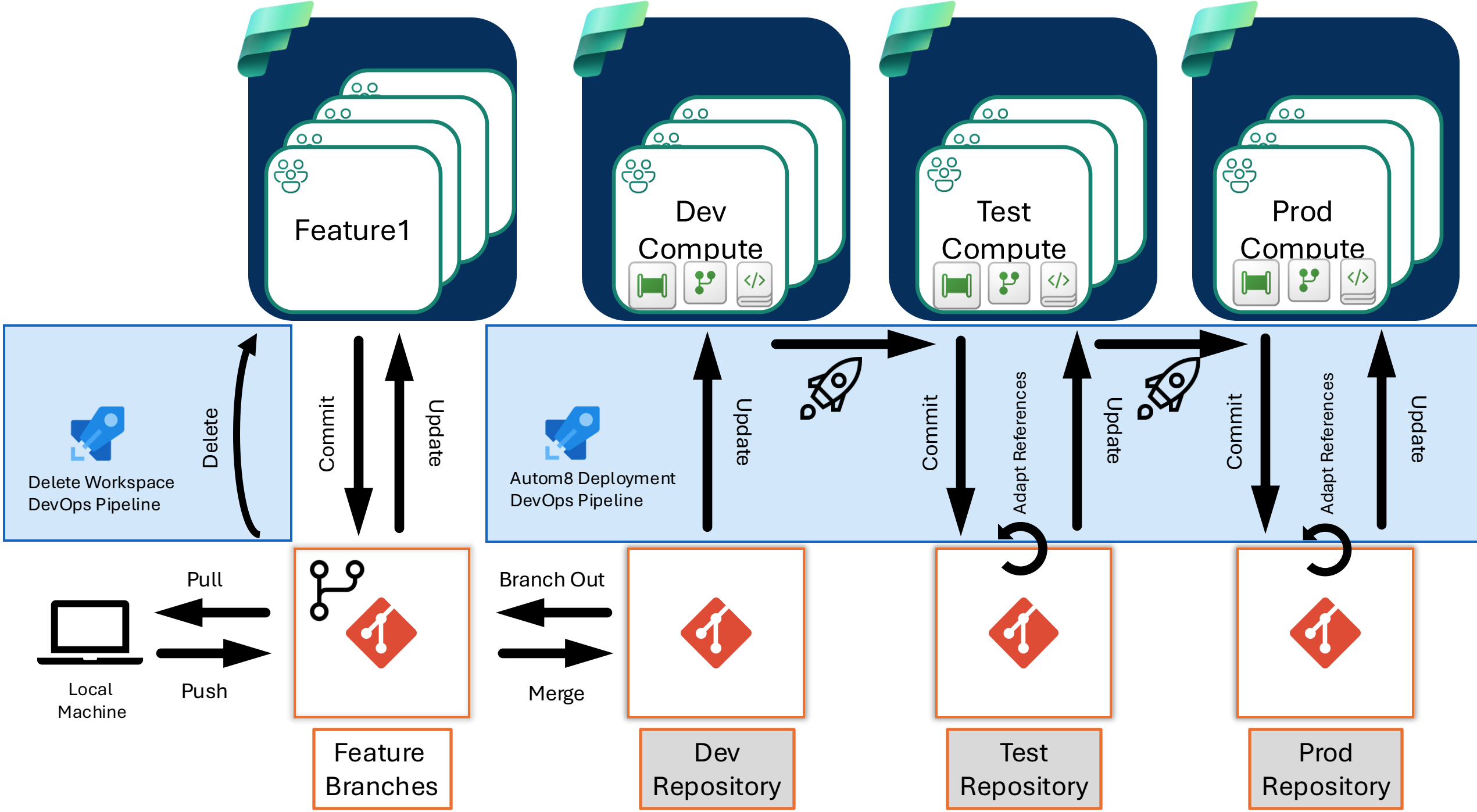
Version control for all workspaces (Dev, Test, Prod)

Adapting References (Dev Lakehouse -> Test Lakehouse)

A pull request should trigger a deployment pipeline automatically

Workspaces should be synced with Git automatically

Repositories
Dev
Test
Prod
Storage
Storage
Storage
/Storage
/Compute
Compute
Compute
Compute
/Compute
/Reports
Reports
Reports
Reports
Deployment Pipelines



Compare to: https://data-marc.com/2024/07/09/fabric-ci-cd-with-Git-deployment-and-release-strategies/

CI/CD in Action

Fixing Git and CI/CD

Fabric Item	Git Integration	Deployment Pipelines
Lakehouses	x (partially)	x
Warehouses	x	x
Data Pipelines	x	x
Notebooks	x	x
Paginated Reports	x	x
Spark Job Definitions	x	x
Spark Environments	x	x
Reports	x	x
Semantic Models	x	x
Dataflows Gen2	x	x
Datamarts	x	x
Dashboards	x	x
Eventhouses	x	x
Eventstreams	x	x
KQL Database	x	x
KQL Queryset	x	x
ML Model	x	x
ML Experiment	x	x

Fixing CI/CD in Microsoft Fabric

fabric_trigger_deployment_pipeline.yml

Edit

Contents History Compare Blame

```yaml
    - task: AzureKeyVault@2
      inputs:
        azureSubscription: 'Visual Studio Enterprise-Abonnement MPN(3bc27d58-dfd4-4148-be
        KeyVaultName: 'kv-stone-boot-vis-001'
        SecretsFilter: 'kerriganpw'
        RunAsPreJob: false

    - task: PowerShell@2
      displayName: Login
      inputs:
        targetType: 'inline'
        script: |
          az login -u sarah.kerrigan@steinaudev.onmicrosoft.com -p $(kerriganpw)
        pwsh: true
```

Non-interactive user login

One needs to **compromise the security** of **the entire Azure tenant** for this to work. Do not try at home.

Service Principals must work for all Endpoints

Best Practices

- Develop on feature branches
- Use branch protection for the main branch
- Do small commits
- Learn using Git properly
- Use Service Principals for automation

Thank you for your attention.
Any questions?