Repositories and Tools

Dr. Chuck Easttom

Case 2

The PATRIOT is a surface to air missile system used by the US Army. PATRIOT is an acronym for Phased Array Tracking Radar to Intercept On Target. On Feb 25, 1991, on Iraqi Scud missile hit a barracks in Saudi Arabia killing 28 US soldiers. The investigation revealed the PATRIOT failure was due to a software error in the systems clock. The missile battery had been in operation for 100 hours, by that time the systems internal clock had drifted by 1/3 of a second. This led to a miss of 600 meters.

What is a Repository? (Repo)

- A collection of all the files and the
- history of those files.
- which consist of all your commits.
- **A place where all your hard work is stored.**

Version Control System

Version control is important when you're working on a software project over time.

- Many source and documentation files.
- The files change during development.
- You need to keep track of the different versions of each file and possibly roll back to an earlier version.

A version control system (VCS):

- Records changes to your files over time.
- You can recall specific versions later.

Version Control System

A VCS is critical for a multi-person project.

How do you keep developers on a team from overwriting each other's changes?

How do you roll back to an earlier version of a file after deciding that the latest changes were bogus?

Who has the latest and greatest version of the source files?

Common terminologies

Repository (Repo): This is where your code lives. It's a centralized storage space for all code files, revision history, and more. Think of it as a bookshelf dedicated to a specific story. The shelf doesn't just hold the latest version of the story but all its drafts and changes too.

Commit: When you make changes to your code and are satisfied, you "commit" those changes, effectively taking a snapshot of your code at that moment. It's like finishing a chapter in your story and taking a photo of it.

Branch: Sometimes, developers need to create a separate environment from the main project to work on new features or experiments without affecting the main project. Imagine creating a spin-off from the main story. You branch out to write a side story without changing the main plotline. Once you're satisfied with the spin-off, you can integrate it back into the main story.

Merge: Once the work on a branch is complete and tested, it can be combined (or "merged") back into the main project. Once the side story (branch) has been finished and polished, you can weave it back into the main story.

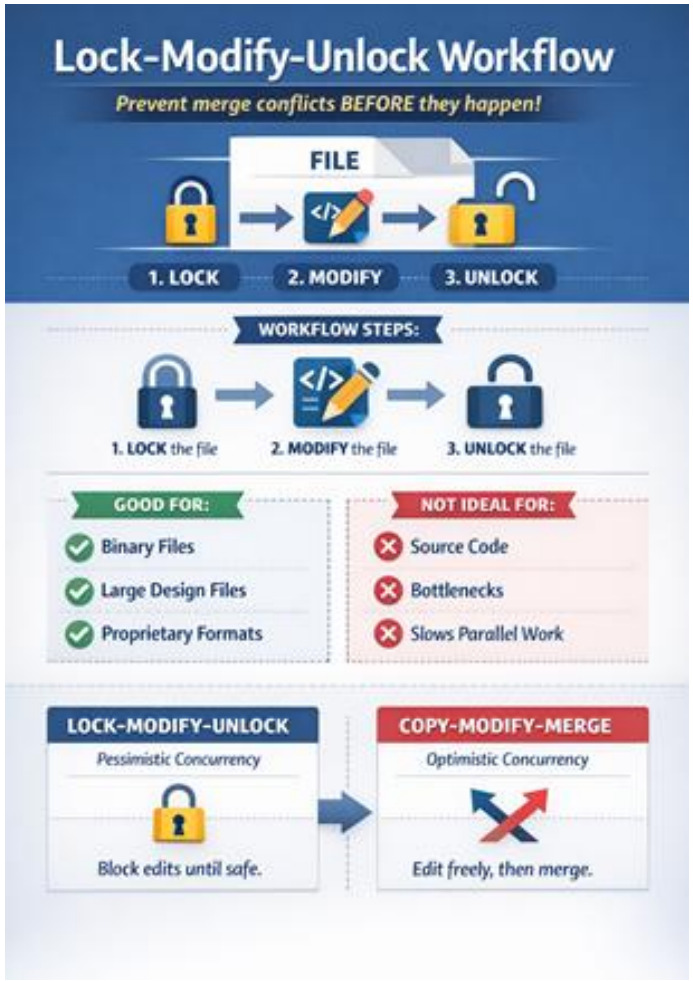
Sehgal, Vandana Verma. Implementing DevSecOps Practices: Understand application security testing and secure coding by integrating SAST and DAST . Packt Publishing.

The Lock-Modify-Unlock Solution

Many version control systems use a *lock-modify-unlock* model to address this problem The **lock-modify-unlock** model is a way of working in a source code repository where a developer:

- **Locks** a file so others cannot edit it at the same time.
- **Modifies** the file locally.
- **Unlocks** it after committing or checking in the change.

The main goal is to prevent merge conflicts before they happen.
In practice, it works like this:
You mark a file as reserved for editing.
The repository records that reservation.
Other developers can still read the file, but they are blocked, or strongly warned, from changing it until the lock is released.



The problem of lock-modify-unlock model

Locking may cause administrative problems. Sometimes an individual will lock a file and then forget about it. Meanwhile, another developers is still waiting to edit the file,

Locking may cause unnecessary serialization. What if the person is editing the beginning of a text file, and another team member simply wants to edit the end of the same file? These changes don't overlap at all. They could easily edit the file simultaneously, and no great harm would come, assuming the changes were properly merged together. There's no need for them to take turns in this situation.

Locking may create a false sense of security. Locking does not eliminate all risk.

The Copy-Modify-Merge solution

Subversion, CVS, and other version control systems use a *copy-modify-merge* model as an alternative to locking. In this model, each user's client reads the repository and creates a personal ***working copy*** of the file or project. Users then work in parallel, modifying their private copies. Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a human being is responsible for making it happen correctly.

Conflicts

But what if one developers changes *do* overlap with another developers changes? What then? This situation is called **a *conflict***, and it's usually not much of a problem. Note that software can't automatically resolve conflicts; only humans are capable of understanding and making the necessary intelligent choices. Once the user has manually resolved the overlapping changes he or can safely save the merged file back to the repository.

The advantage of copy-modify-merge model

The copy-modify-merge model may sound a bit chaotic, but in practice, it runs extremely smoothly. Users can work in parallel, never waiting for one another. When they work on the same files, it turns out that most of their concurrent changes don't overlap at all; conflicts are infrequent. And the amount of time it takes to resolve conflicts is far less than the time lost by a locking system.

Trunk-Based Development

How it works:

Developers commit frequently to a shared main branch

Use **short-lived branches or none at all**

Heavy use of CI/CD and automated testing

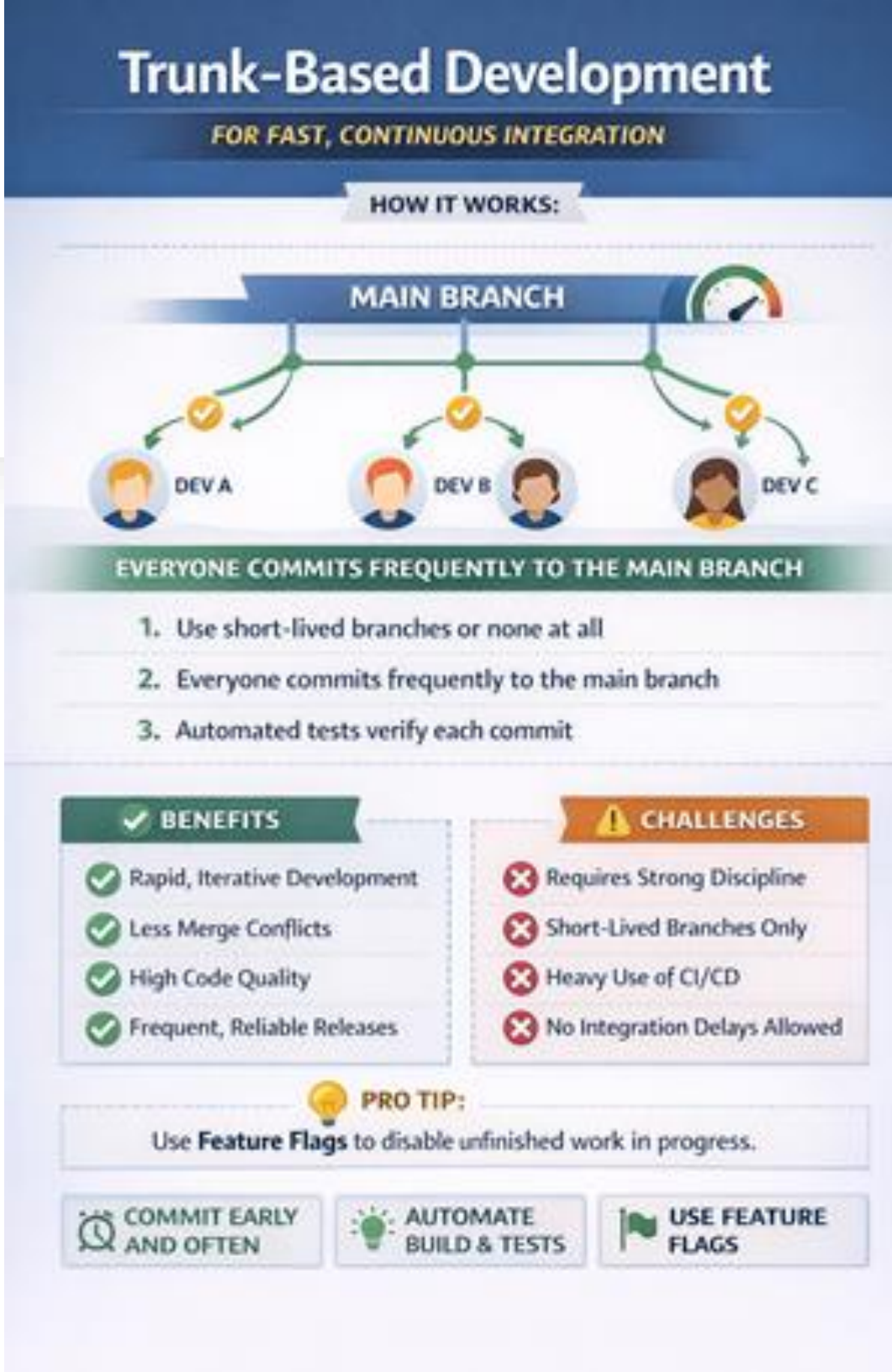
Pros:

Reduces long-lived merge conflicts

Faster integration

Cons:

Requires strong discipline and testing



File-Level or Advisory Locking (Hybrid approach)

)

Used in tools like Git LFS or Perforce.

How it works:

Most files use copy-modify-merge

Certain files (e.g., binaries) use **optional locking**

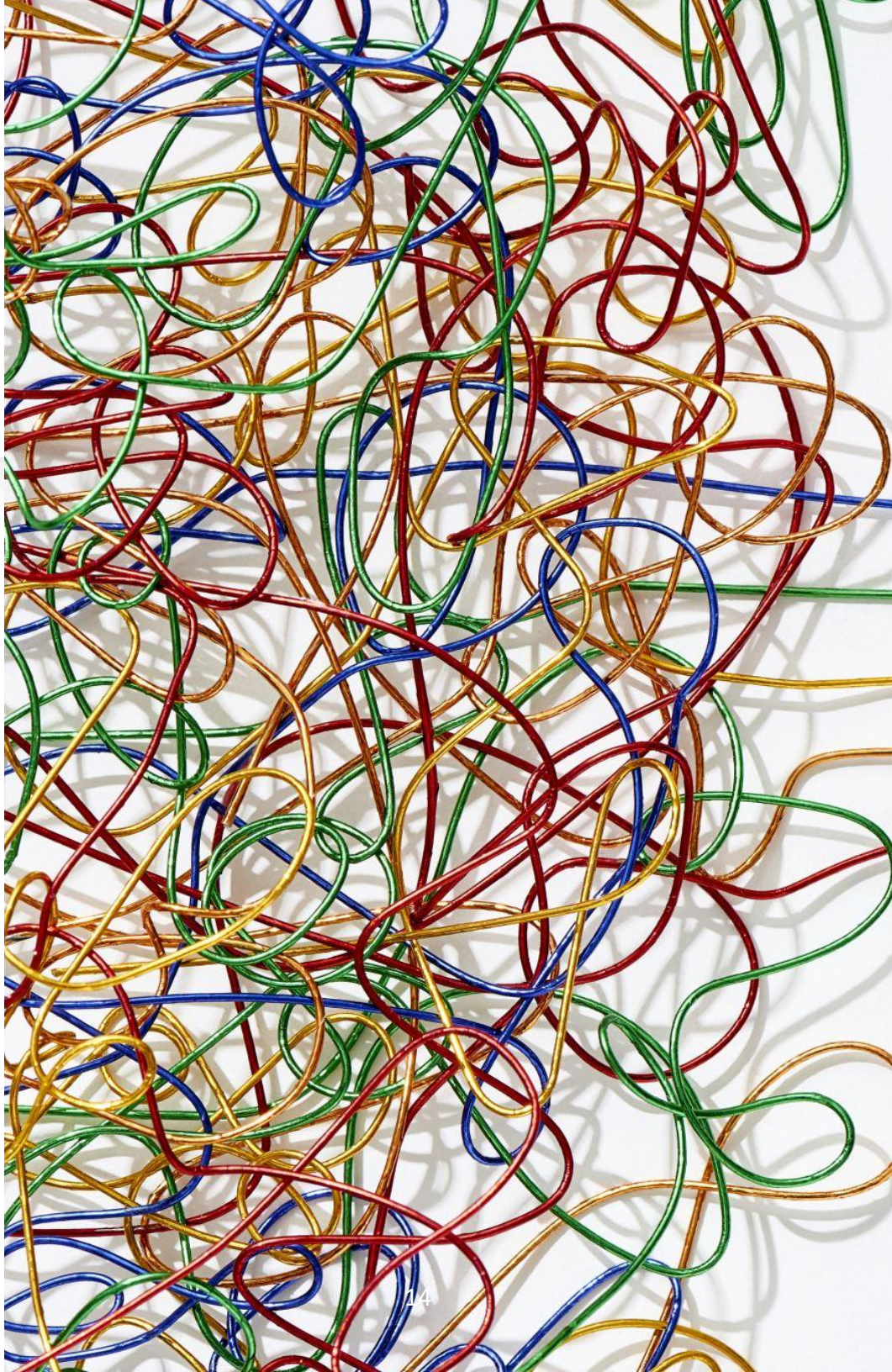
Pros:

Best of both worlds

Avoids conflicts where merging is impossible

Branching / Tagging

One of the features of version control systems is the ability to isolate changes onto a separate line of development. This line is known as a *branch*.



Merging

Where branches are used to maintain separate lines of development, at some stage you will want to merge the changes made on one branch back into the trunk, or vice versa.

It is important to understand how branching and merging works in Subversion before you start using it, as it can become quite complex.

What is Git?

- A tool which allows you to manage and track changes to files over time.
- When you create a repository you will see a .git directory.
- It keeps track of a file's history – its tracks changes and who made those changes.
- Each version of a file is called a commit.

Git

Originally developed by Linus Torvalds.

- Creator of Linux.
- "I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git."
  - git = British term for a silly or worthless person

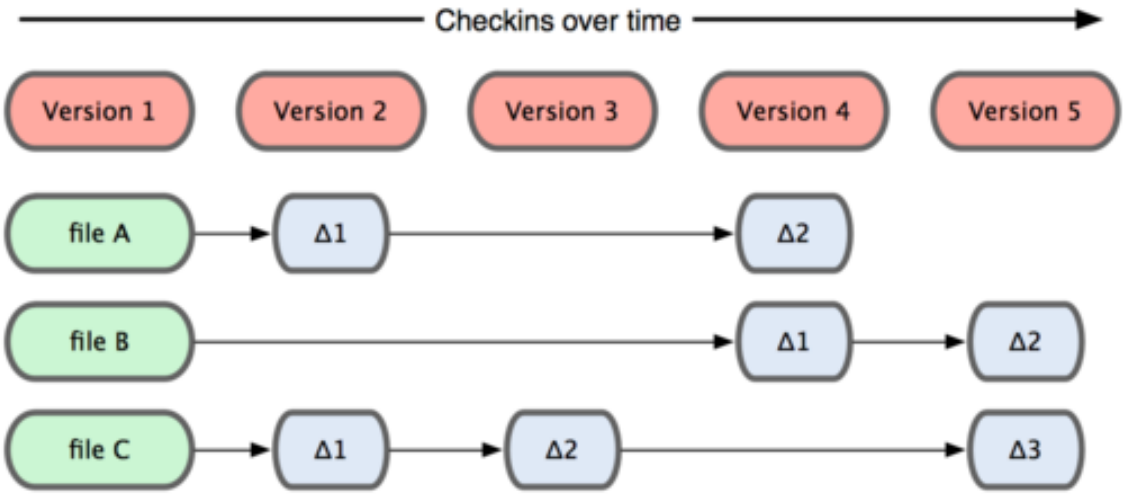
"Global Information Tracker"

Extremely popular in industry and academia.

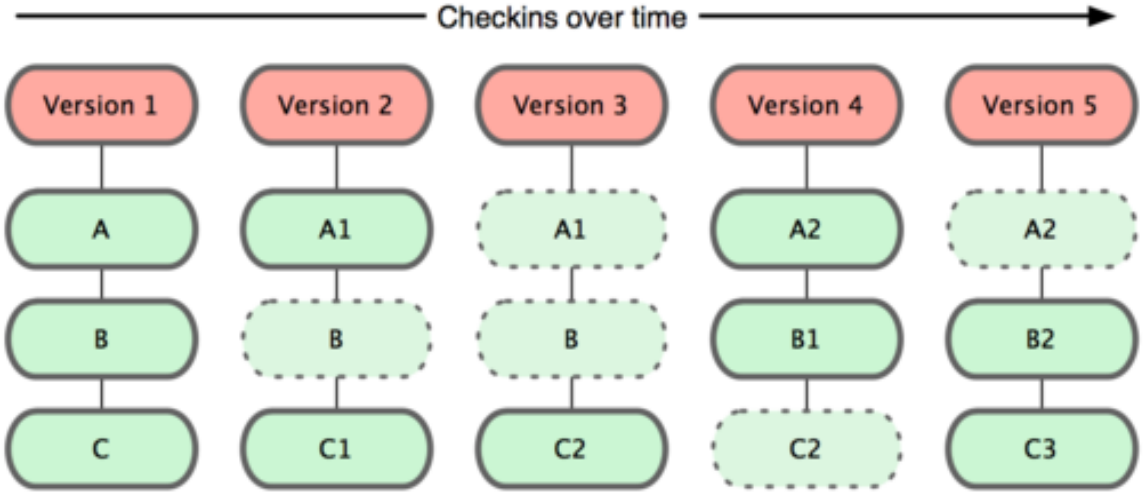
- For example, used by NASA.

Git: File Differences vs. Snapshots

Differences (Subversion)



Snapshots (Git)



Git: Local Operations

Each developer has a
complete local copy of the repository
on his or her laptop or workstation.

Local operations

- add
- commit
- check in
- check out
- status
- differences
- etc.

Git: Remote Operations

Do remote operations only when necessary to the master repository on the server.

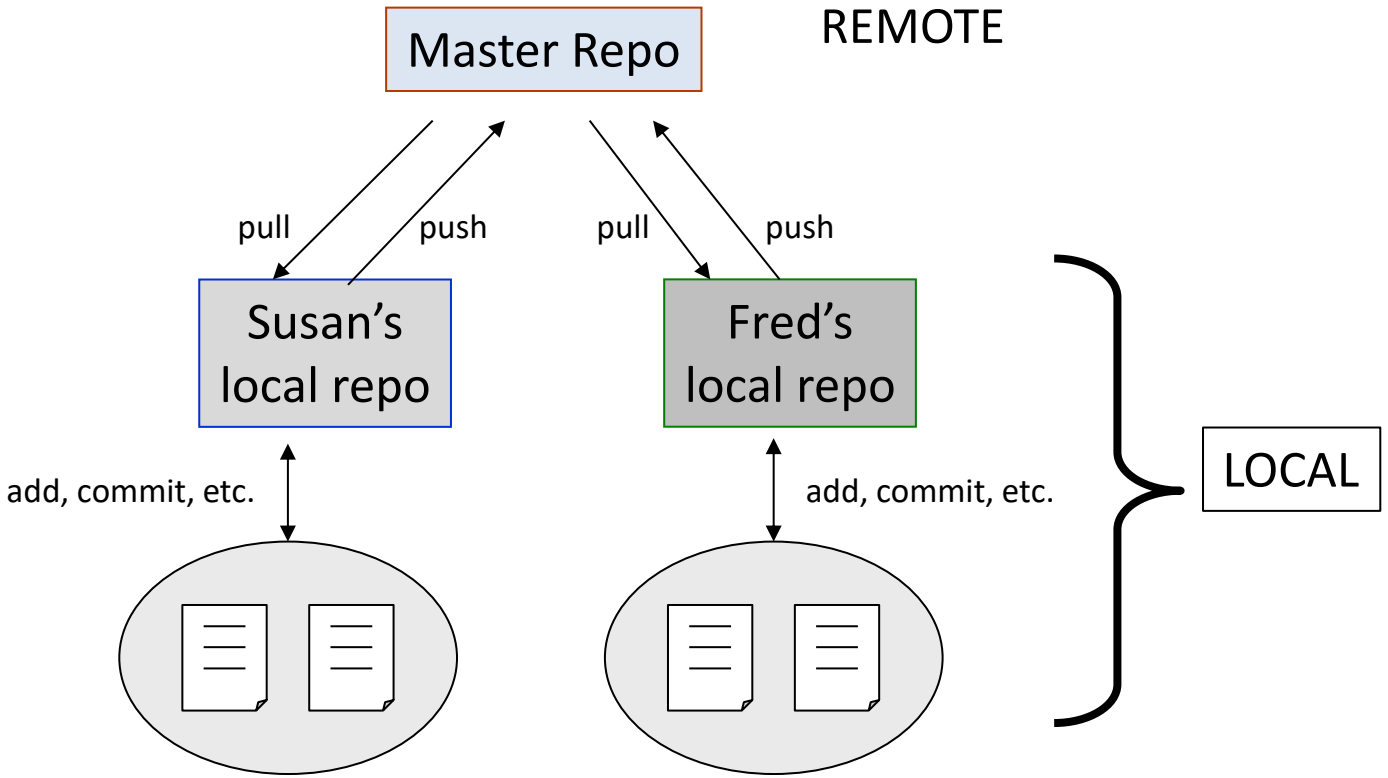
Clone the master repository into a local repository.

Push (and merge) a local repository up to the master repository.

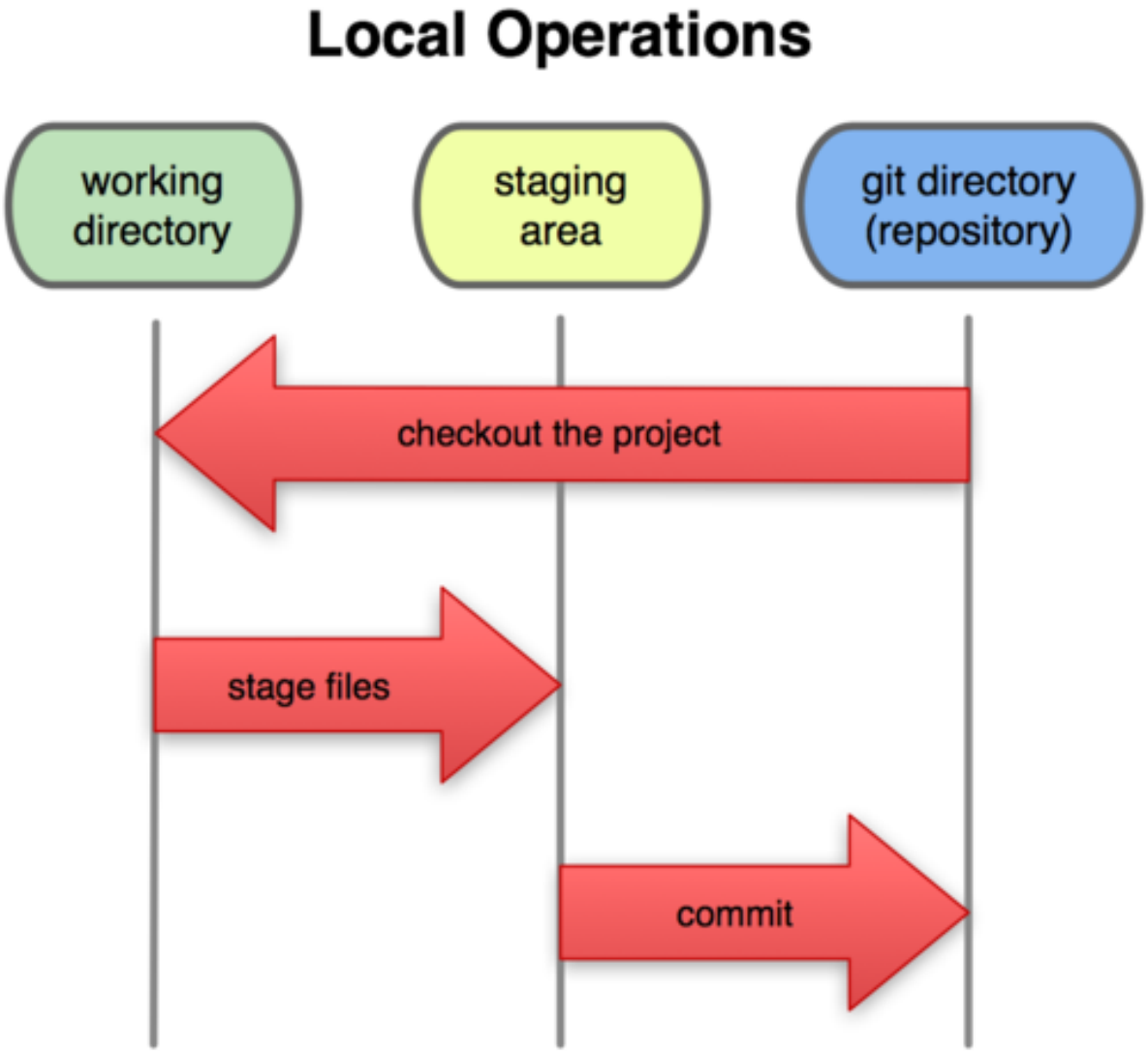
Pull (and merge) files from the master repository down to a local repository.

Git can work peer-to-peer without a master repository.

Local and Remote Repositories



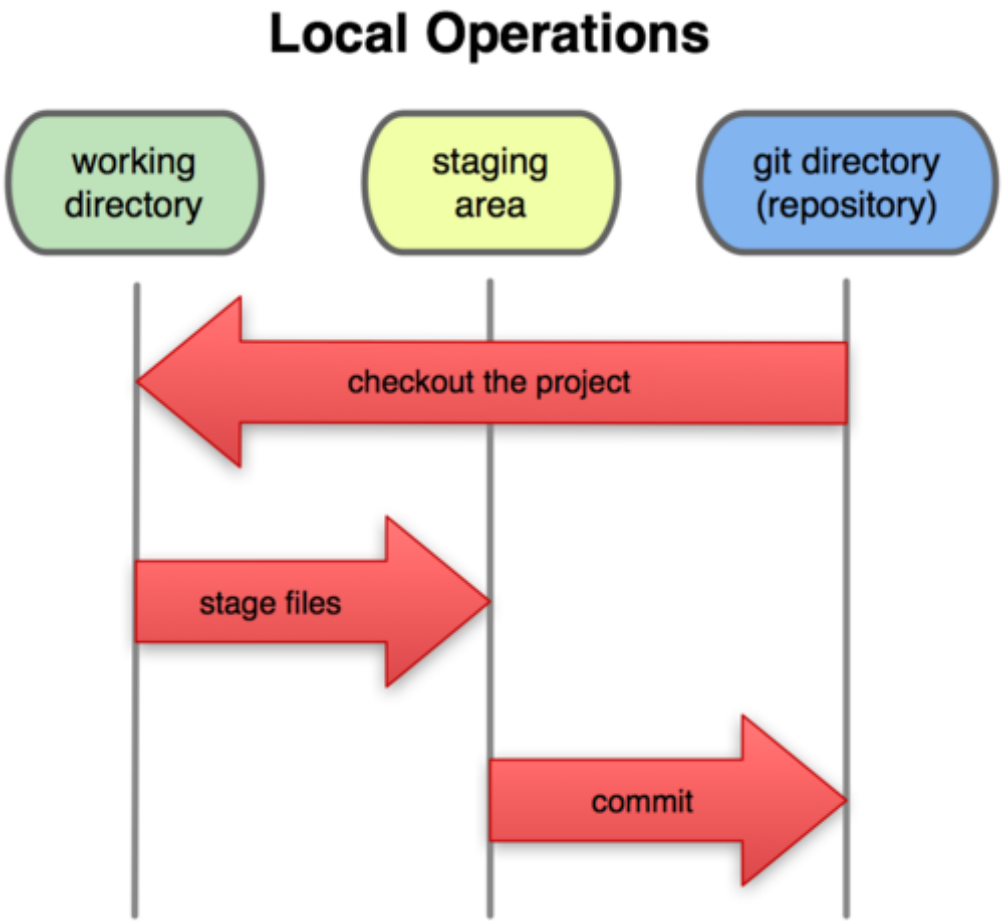
Git: Three File States



A file can be in one of three states:

- modified
- staged
- committed

Git: Three File States



The staging area is also known as the "index".

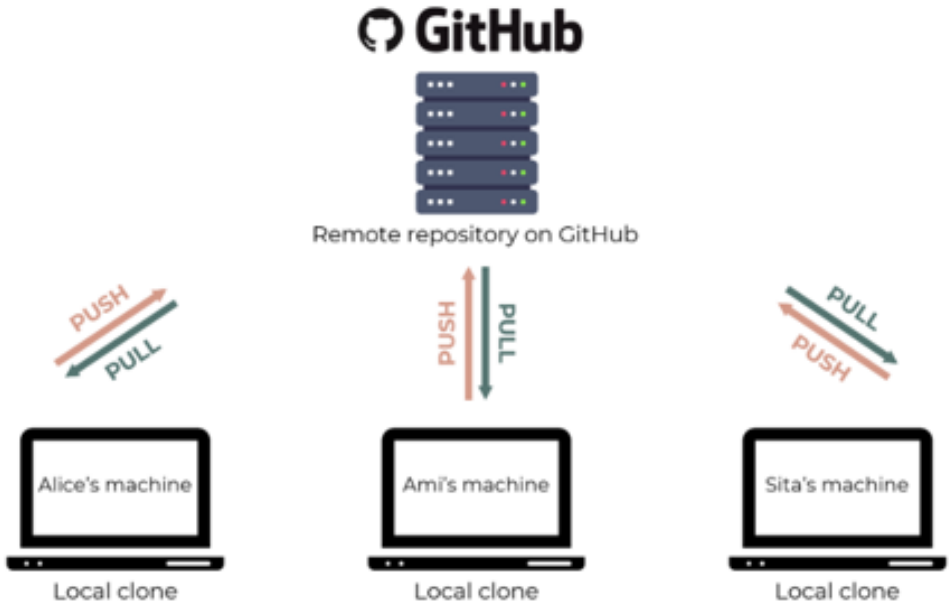
The hidden Git directory `.git` is the local project repository.

The working directory is where you check out a version of the project for you to work on.

The staging area is a file that keeps track of what will go into your next commit.

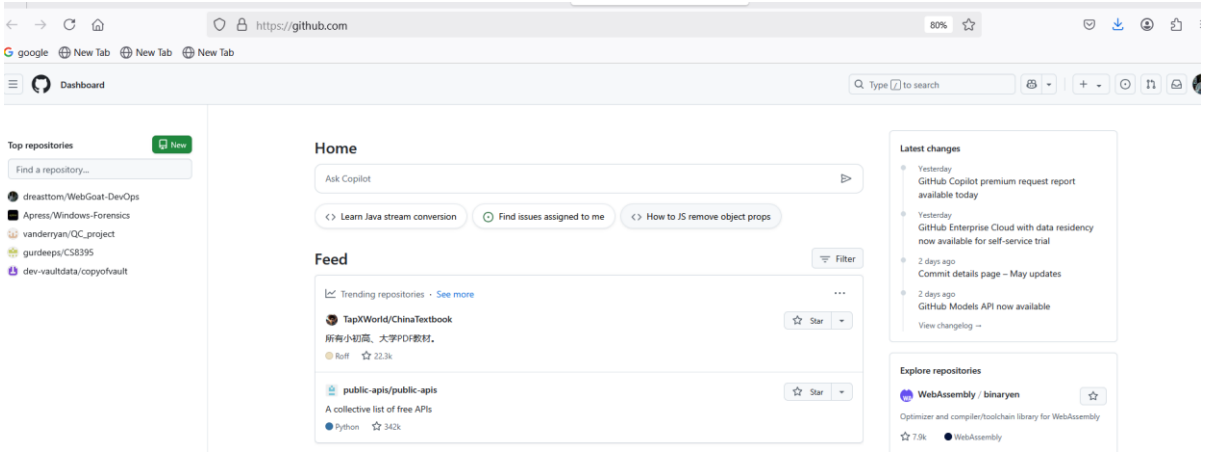
What is GitHub?

- A remote repository in which you can host your local repository.
- **And a place to collaborate with others.**



GitHub

GitHub is a popular website for hosting remote Git repositories. https://github.com/ The free service hosts public repositories.



GitHub: Create Personal Account

Go to http://git-scm.com/downloads

Download and install Git on your local machine.

Go to https://github.com/

Create a personal account with a recognizable user name such as johndoe or jdoe.

Set up SSH keys to establish a secure connection with GitHub.

See https://help.github.com/articles/generating-ssh-keys

GitHub: Create Team Repository

Obtain the GitHub username
of each team member.

Add team members as contributors
to the team repository.

- Click Settings in the right side panel.
- Click Collaborators in the left Options panel.
- Add collaborators using each team member's GitHub username.

GitHub: Create Local Repository

Each team member creates local repository that is a clone of the master repository.

Log into your personal GitHub account.

Search for your team's master repository by name
in the search box at the top.

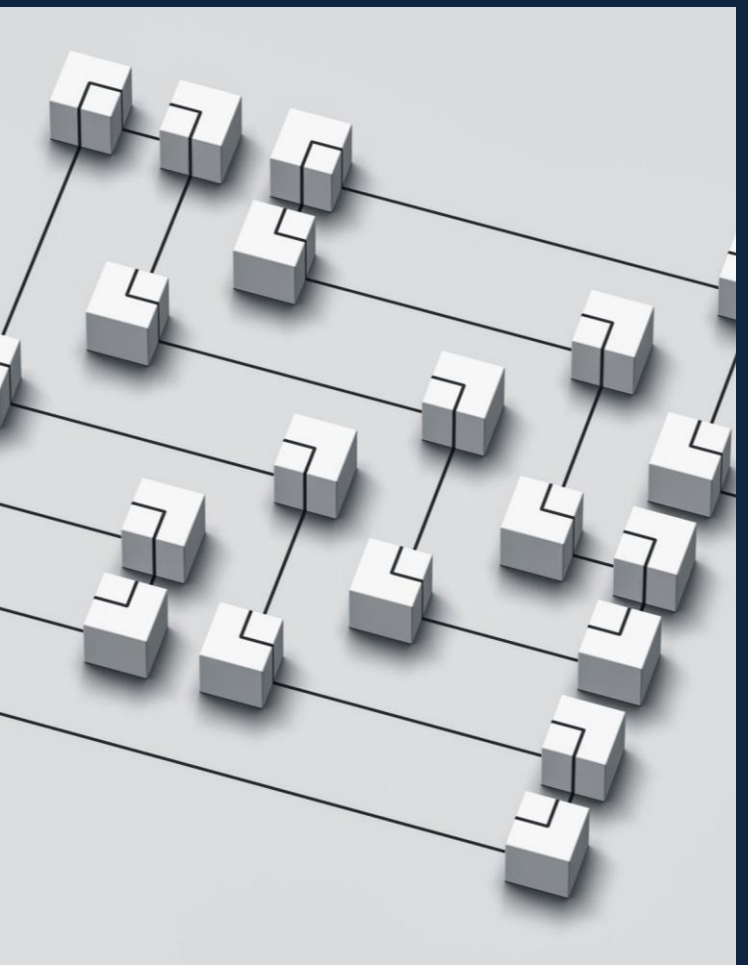
Click on the link to the team repository.

Obtain the URL to the team repository.

On the team repository page, look for HTTPS clone URL in the right-side panel.

Put the URL on the clipboard.

GitHub-Deployment Types



GitHub Free

- Intended for individual developers and small teams
- Public and private repositories
- Basic collaboration tools (issues, pull requests)
- Limited security features

GitHub Team

- Designed for growing teams
- Advanced collaboration features
- Team-level access controls
- Basic security scanning

GitHub Enterprise Cloud

- Enterprise-grade plan hosted by GitHub
- Advanced security and compliance features
- Centralized administration and policy enforcement
- Includes GitHub Advanced Security (optional add-on)

GitHub – Detecting Secrets

How it works

GitHub scans commits, pull requests, and repository history for known secret patterns.

It uses:

- **Pattern matching** (regular expressions)
- **Entropy checks** (to detect high-randomness strings)
- **Contextual validation** (format + surrounding code context)

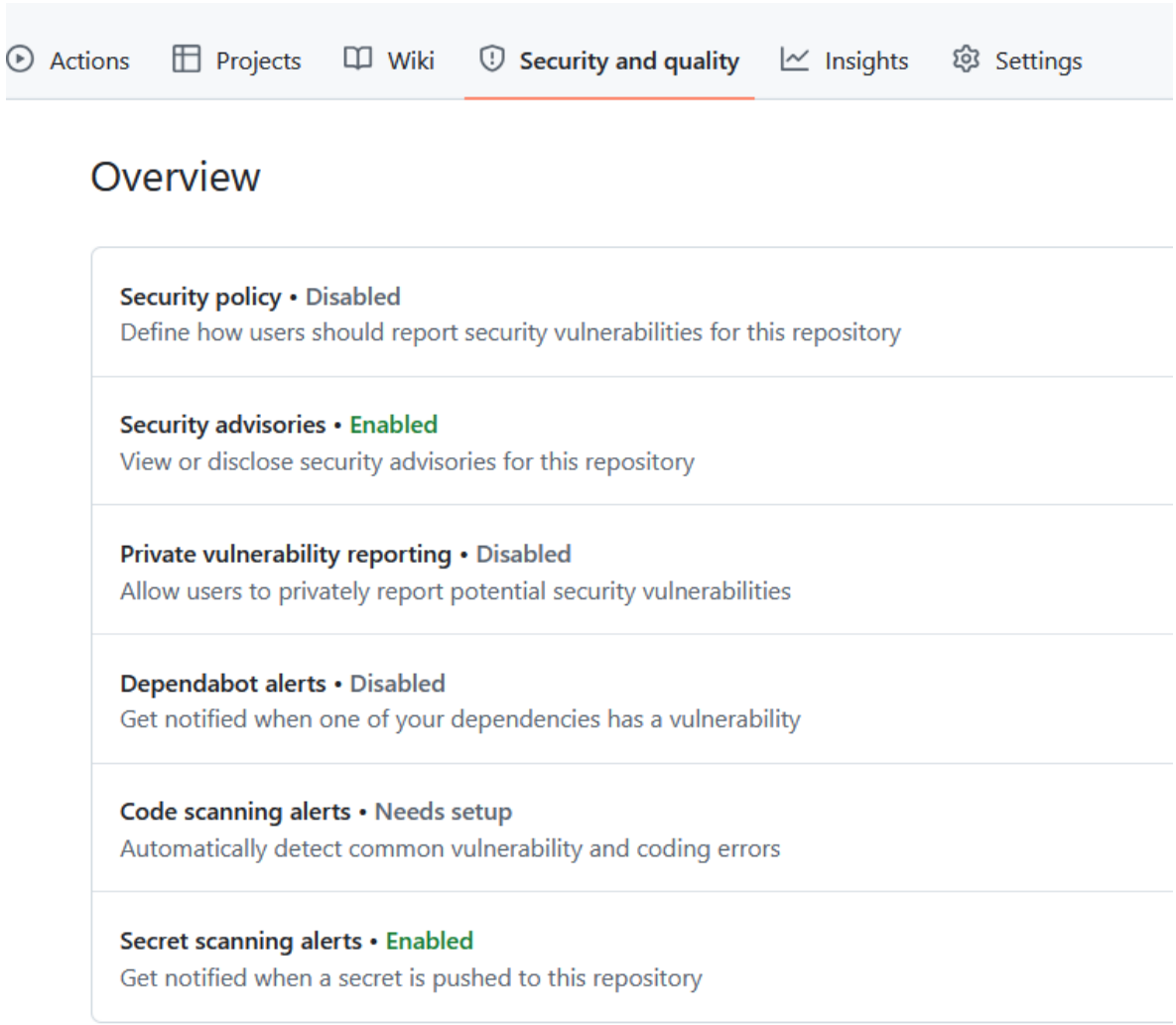
What it detects

Tokens and keys **from hundreds of service providers**, including:

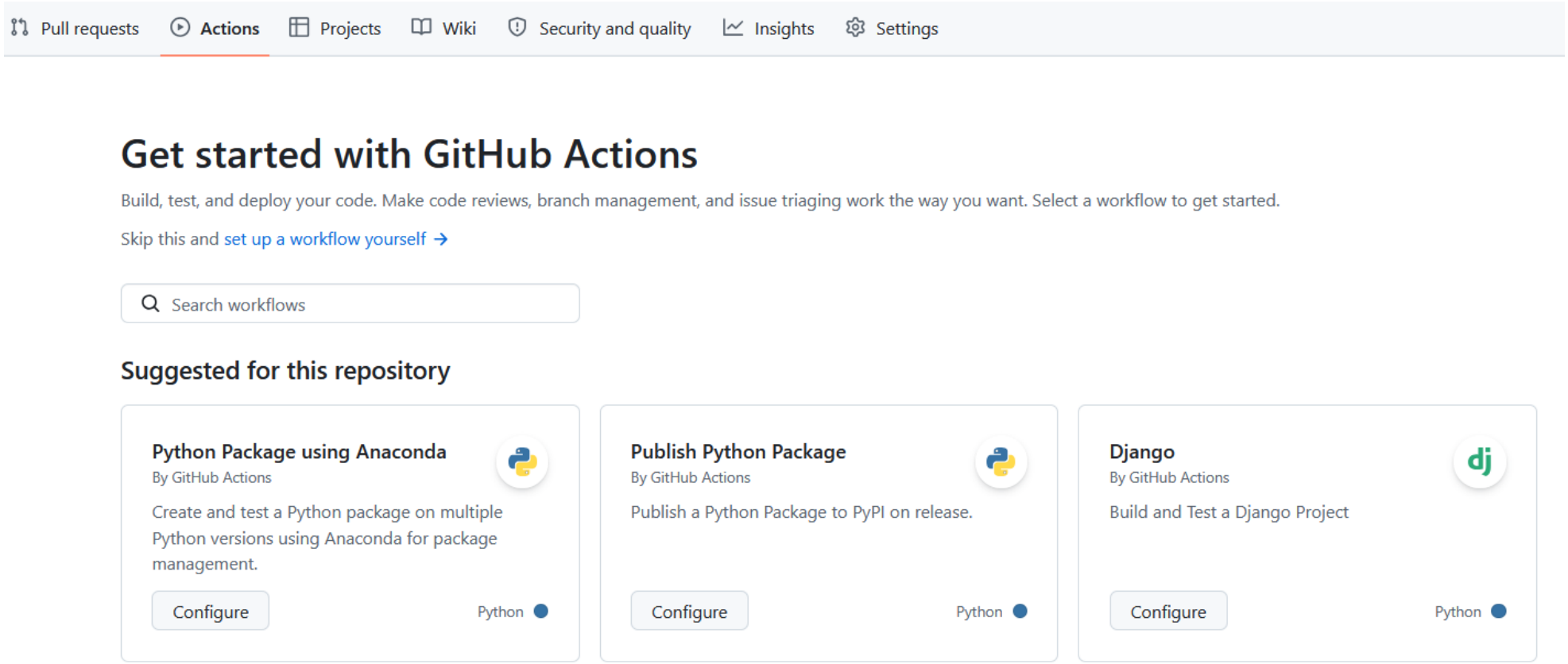
- AWS access keys
- Azure keys
- Google Cloud credentials
- GitHub tokens
- Slack, Stripe, Twilio, GitLab, and many others

GitHub allows organizations to define custom secret patterns.

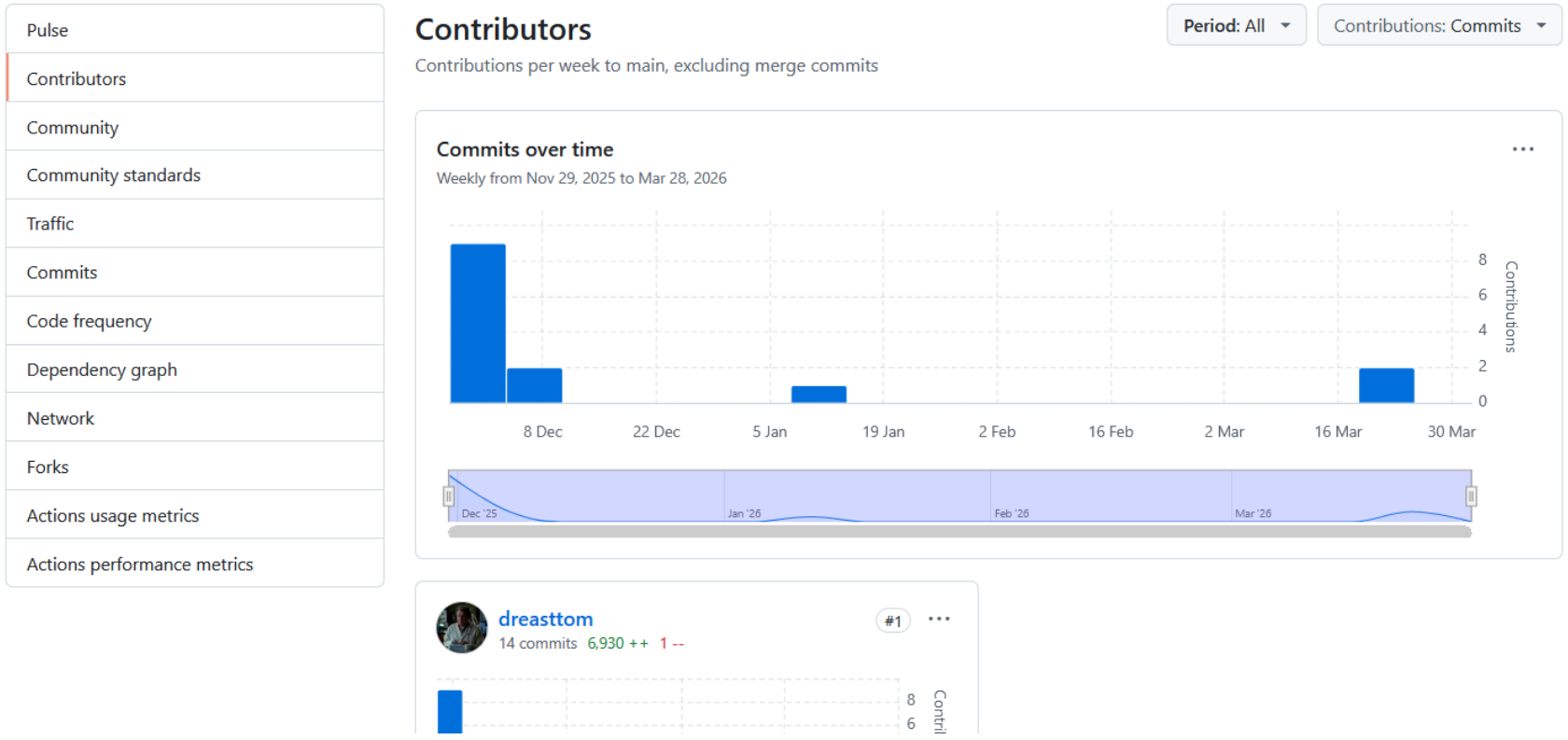
GitHub Security Settings



GitHub Actions



GitHub Insights



GitHub -GraphQL

GraphQL is a query language and runtime for APIs that enables clients to request exactly the data they need, and nothing more, using a single, strongly typed schema. Instead of relying on multiple fixed endpoints, clients send flexible queries that specify the shape of the response, allowing efficient data retrieval and reducing over- or under-fetching. GraphQL supports both reading and modifying data through queries, mutations, and subscriptions, and it provides powerful features such as introspection, version-free API evolution, and precise error reporting. By decoupling client requirements from server implementation details, GraphQL improves developer productivity and makes APIs easier to evolve and consume across web, mobile, and backend systems.

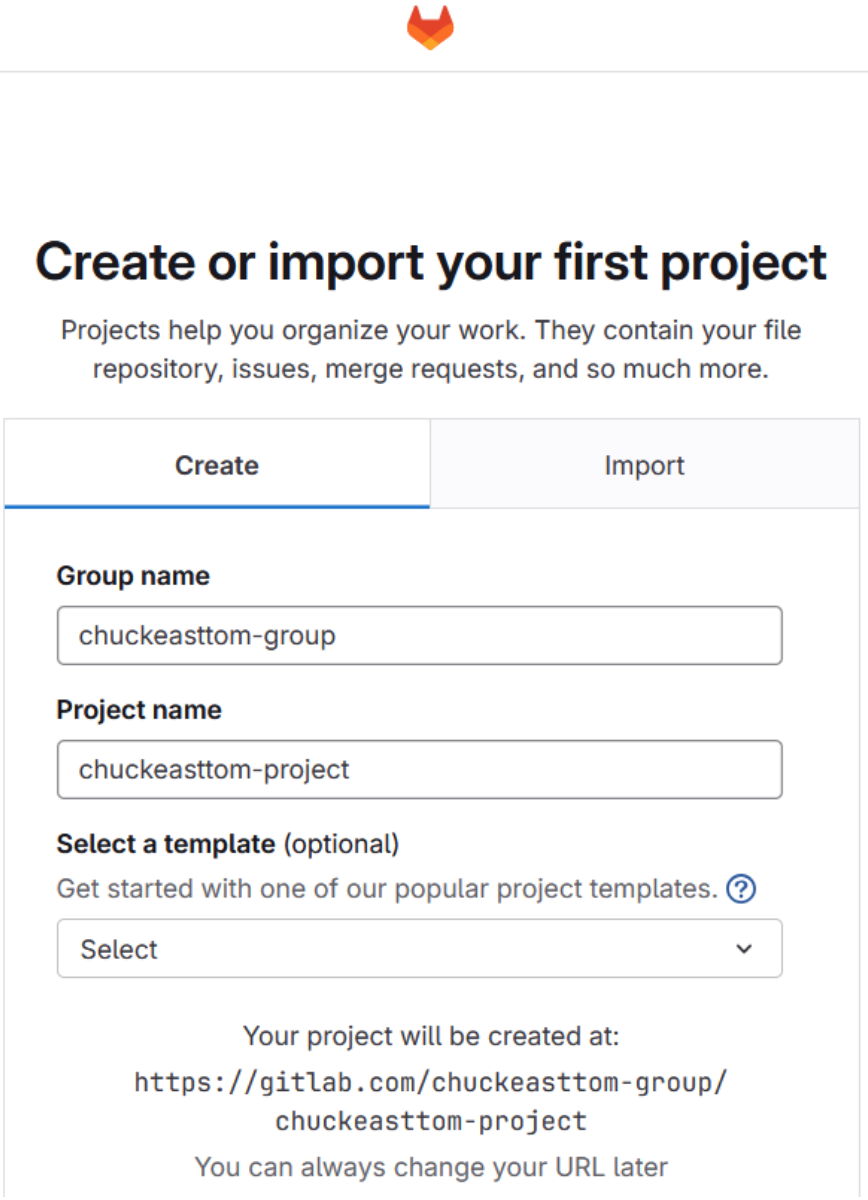
GitHub SCIM vs Team Synch

Feature	SCIM	Team Sync
Primary purpose	User lifecycle management	Team membership management
Scope	Enterprise / organization-wide	Organization teams
Source of truth	Identity Provider (IdP)	Identity Provider groups
Manages users	Create, update, deactivate	No
Manages teams	No	Yes
Requires SSO	Yes	Yes
Typical IdPs	Azure AD, Okta, Ping	Azure AD, Okta, Ping

Note: SCIM (System for Cross-domain Identity Management) is an open standard used to automate the lifecycle management of user identities across systems.

GitLab

- Access site with username and password
- Password must be changed on 1st login



GitLab

Source Code Management (SCM)

- Git-based repositories with support for branches, tags, and protected branches
- Merge requests with inline code reviews and approvals
- Built-in issue references, commit linking, and history tracking
- Fine-grained access controls and audit logs

CI/CD (Continuous Integration & Continuous Delivery)

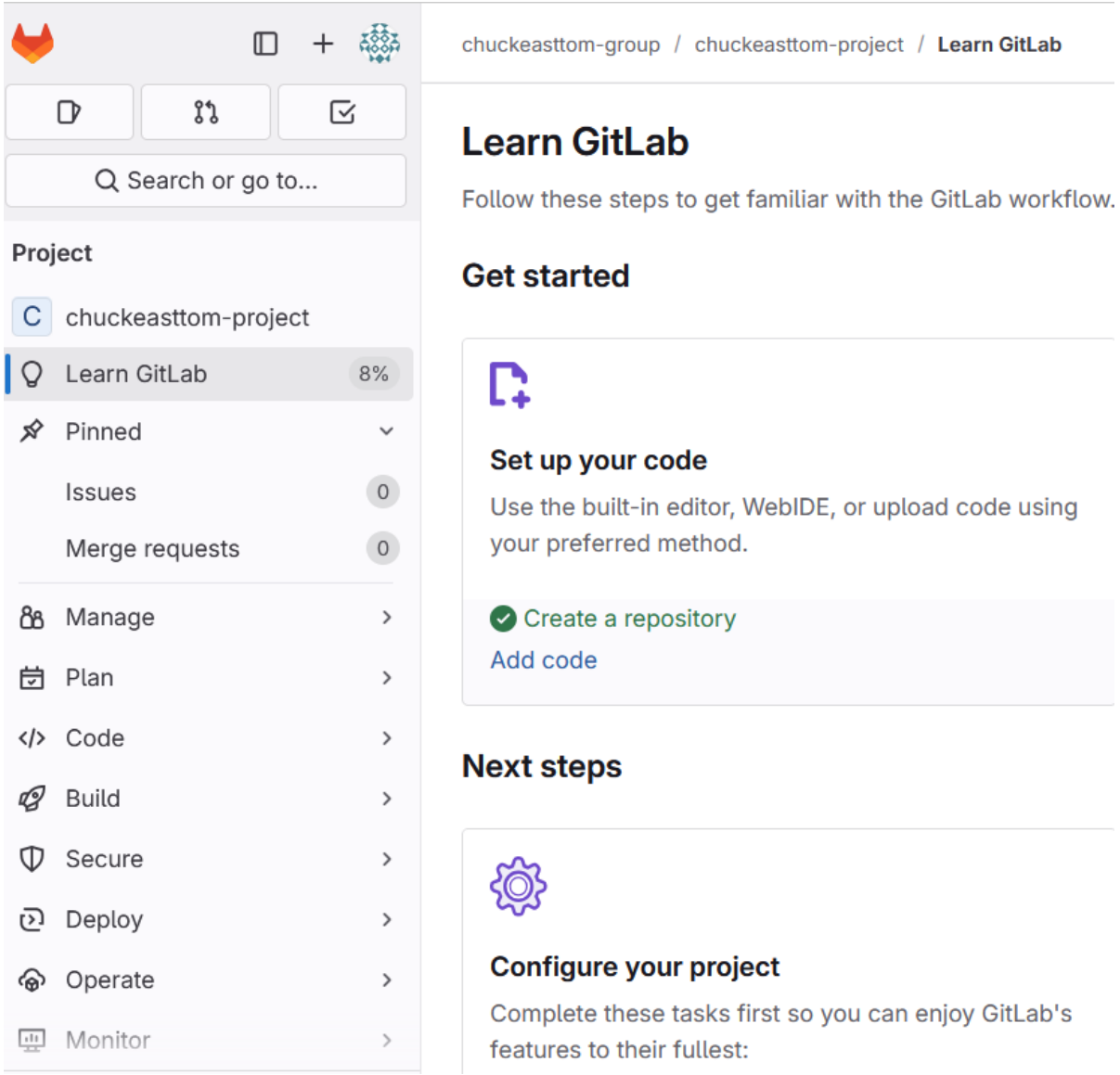
- Built-in CI/CD pipelines defined using .gitlab-ci.yml
- Shared, group, or project-level runners
- Pipeline visualization, job dependencies, and caching
- Automated testing, building, and deployment workflows
- Support for cloud, container, and on-prem deployments

DevSecOps & Security

- Integrated security scanning throughout the pipeline:
  - Static Application Security Testing (SAST)
  - Dynamic Application Security Testing (DAST)
  - Dependency and container scanning
  - Secret detection
- Security dashboards and vulnerability management
- Policy enforcement and compliance reporting

GitLab Console

- Console show activity on repo, projects, settings



GitLab Actions from the Console

- Create projects
- View files
- Add files
- View commit messages
- Handle pull request for merge

GitHub v GitLab

Aspect	GitHub	GitLab
Core focus	Best-in-class **code collaboration platform**	**All-in-one DevSecOps platform**
Tooling model	Ecosystem-based (best-of-breed integrations)	Single application, single data model
Adoption pattern	Developer-first, extensible	Platform-first, standardized

GitHub v GitLab

Feature	GitHub	GitLab
Repositories	Git-based	Git-based
Code reviews	Pull Requests	Merge Requests
Inline comments	Yes	Yes
Protected branches	Yes	Yes
Open-source ecosystem	Industry-leading	Strong, but smaller

GitHub v GitLab

Capability	GitHub	GitLab
SAST	Via Advanced Security	Built-in
DAST	Via Advanced Security	Built-in
Dependency scanning	Via Advanced Security	Built-in
Secret detection	Built-in + push protection	Built-in
Security dashboards	Enterprise-focused	Unified platform view
Policy enforcement	Strong	Strong

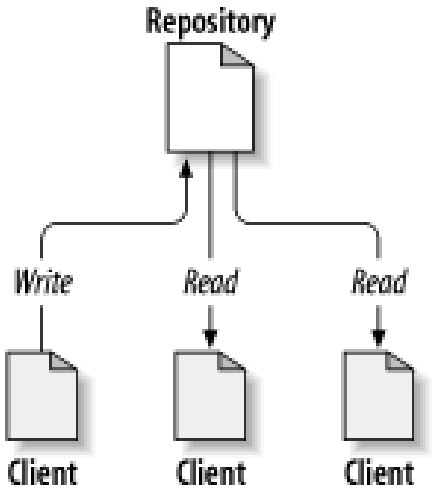
Capability	GitHub	GitLab
SAML / SSO	Yes	Yes
SCIM provisioning	Yes	Yes
Team/group sync	Yes	Yes
Managed users	Yes	Limited
Audit logs	Strong	Strong

GitHub v GitLab

SVN Version Control

- svn is a tool for version control of files
- Keeps a repository of what is under control at a central location
- Allows users to make local (aka working) copies of the repository on their machine or file system
- Saves each version a user commits
- Administrator sets up main repository.

The Repository



Subversion is a centralized system for sharing information. At its core is a *repository*, which is a central store of data.

Getting and Updating

Checkout a repository. A local/working copy of a repository is created on a machine or file system. This links a given folder to the address of the repository.

```
svn checkout address folder
```

Update a local/working copy. The contents of the repository are copied to the local folder. This updates the local copy.

```
svn update
```

Committing Changes

Commit a change. Copy a local change to repository. This commits a change to the repository. This is a new version of the item under version control. It is numbered and the old version is saved. Need to commit:

Modified files

Added or removed files or folders

`svn commit` *file* `-m` "*Commit note*"

Other Essential Commands

`svn add` – add a file. A commit must be done after the add to commit it to the repository.

`svn delete` – delete a file, followed by a commit.

`svn log` – get all the commit messages. Use this to roll back to an earlier version.

Rules to Live By (svn)

1

Always update before you start working

2

Always commit after changes

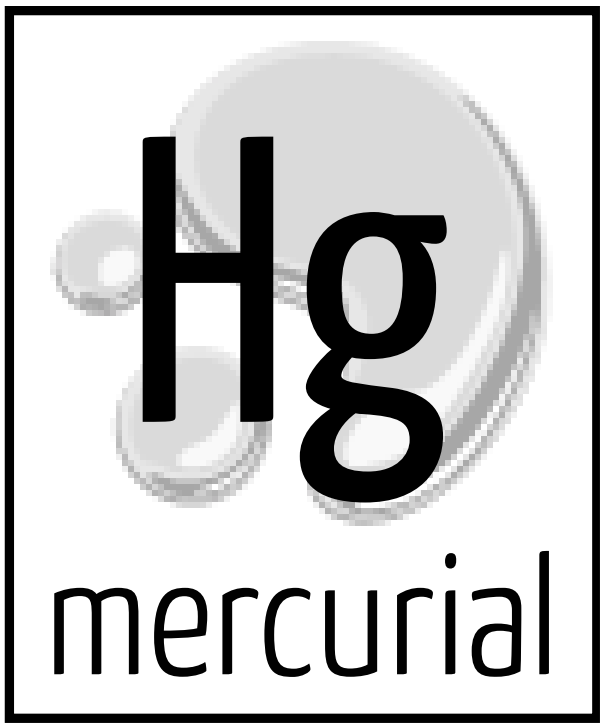
3

Commit often – after a major (or minor) change:

- Add a method
- Fix a bug
- Change a method

Mercurial

Mercurial is a command line tool released in April 2005. Every user has a full copy of the repository, including history. It is designed to be user friendly/simple to use. Commands follow a regular pattern (hg add, hg commit, hg push, etc.). There is TortoiseHg GUI tool for easier visualization and management. Works on Windows, macOS, Linux.



https://www.mercurial-scm.org/

TFS (Team Foundation Server) / Azure DevOps Server

In 2018, Microsoft rebranded TFS to Azure DevOps Server. Supports both Git (distributed) and TFVC (Team Foundation Version Control – centralized). It also includes Agile tools for sprint planning, tracking work items, bugs, etc.

DevOps Server express is a free version for individuals and small teams. Use Azure DevOps Server Express as individual developers or teams of five or less, at no cost. Easily install on your personal desktop or laptop without needing a dedicated server.

https://azure.microsoft.com/en-us/products/devops/server

Fossil

Fossil is a distributed version control system that in d-
in web interface, and project management tools — as
created to support the SQLite project and is known d
self-contained architecture. Can be used with multi

https://fossil-scm.org

SCM (software configuration management)

Centralised SCM

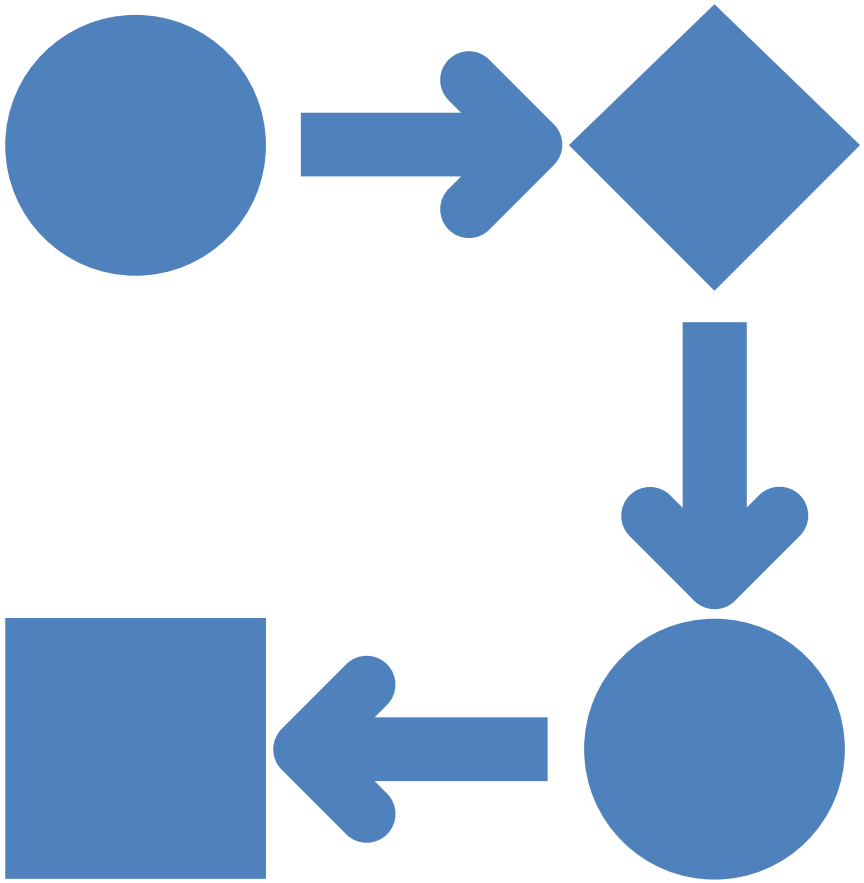
A centralised SCM stores all of its metadata in a single authoritative (or "master") database that is not replicated (except possibly for backup purposes). CVS (and RCS and SCCS) follows this model, as does Subversion.

Distributed SCM

A **distributed SCM** tool (abbreviated: a DSCM tool) is designed to support a model in which each repository is loosely coupled to many others. Each repository contains a complete set of metadata describing one or more projects. These repositories may be located almost anywhere. Individual developers only need access to their own repositories, not to a central one, in order to commit changes.

Distributed SCMs provide mechanisms for propagating changes between repositories.

Distributed SCMs are in contrast to centralised SCMs. Mercurial is a DSCM.



SCA

Software Composition Analysis (SCA) is a process and set of tools designed to identify, manage, and secure open-source components used in a software application. In modern development, where applications are often composed of 60–90% open-source code, SCA plays a crucial role in ensuring software quality and reducing security risks.

“SCA is like a dedicated quality assurance team that covers security and compliance for your software’s ingredients. It ensures you’re building your software with the most up-to-date, safest, and compliant components available. This not only ensures a better end product but also saves you from potential headaches, be they security breaches or legal battles, down the road.”

- Sehgal, Vandana Verma. Implementing DevSecOps Practices: Understand application security testing and secure coding by integrating SAST and DAST (pp. 201-202). Packt Publishing. Kindle Edition.

SCA Process

- **Codebase Scanning**:
  - SCA tools scan source code, binaries, and package manifests (e.g., package.json, pom.xml, requirements.txt).
- **Fingerprinting**:
  - Components are hashed or fingerprinted and compared to large databases of known libraries.
- **Correlation**:
  - Identifies associated metadata: version, license, known CVEs.
- **Reporting & Remediation**:
  - Tools generate reports, prioritize issues, and often suggest fixed versions.
  - Similarly, the SCA tool doesn't just identify problems. It provides recommendations, such as updating a component to a safer version or replacing it with a more secure alternative.
- **Continuous monitoring:**
  - The team doesn't just check ingredients once. They continuously monitor supply chain news and updates. SCA tools also offer continuous monitoring, ensuring that if a vulnerability is discovered in the future in a component you're using, you're notified immediately.

Sehgal, Vandana Verma. Implementing DevSecOps Practices: Understand application security testing and secure coding by integrating SAST and DAST (p. 204). Packt Publishing. Kindle Edition.