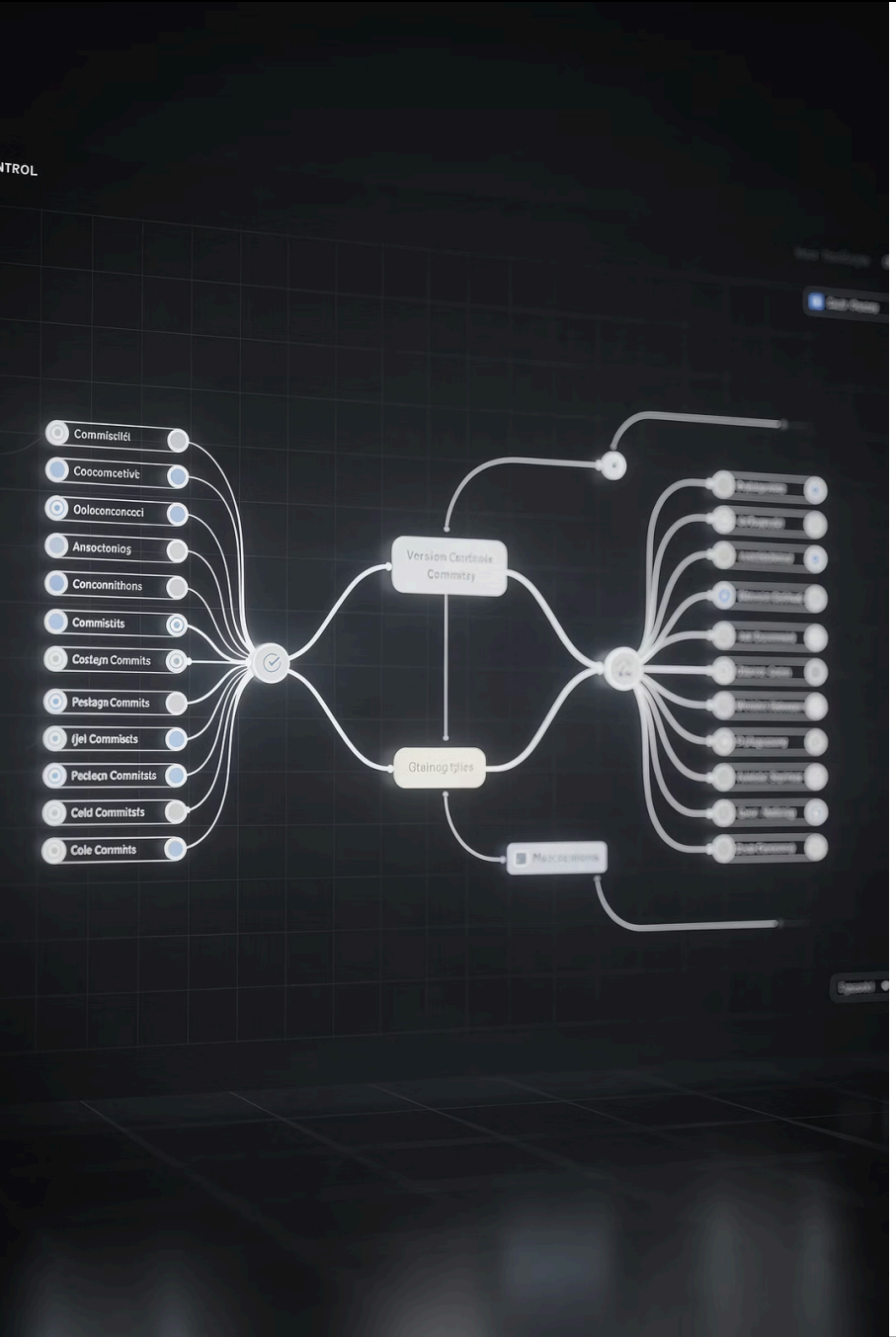
Module 4: Manage Local and Git / Version Control Repositories in AAP

DURATION: 3 HOURS

THEORY: 30% | HANDS-ON: 70%

Designed for Ops / DevOps / Platform Engineers working with **Red Hat Ansible Automation Platform (AAP 2.4/2.5)** — this module bridges automation expertise with Git-based source control practices.



4.1 Introduction to Version Control

Before automation can scale, it needs a foundation. This section establishes **why version control is essential** in enterprise automation environments — and how Git becomes your team's single source of truth.

What Is Version Control and Why It Matters

Version control is a system that tracks changes in files over time. In automation, **code is infrastructure** — and that code changes constantly.

Without Version Control

- **No history tracking**
- **No rollback capability**
- **No team collaboration**
- **High risk of production failure**

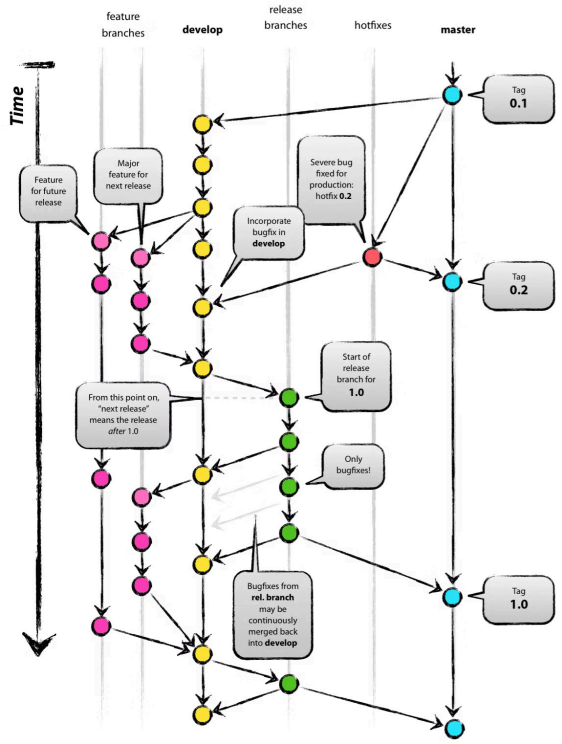
With Version Control

- **Full audit trail**
- **Rollback to any state**
- **Team collaboration**
- **Change approval workflows**
- **Compliance readiness**

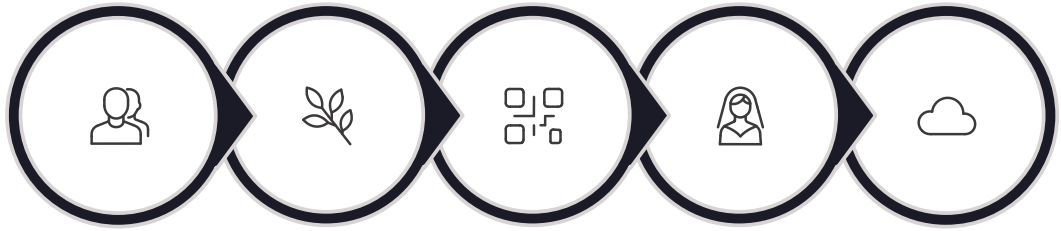
In enterprise automation — covering playbooks, inventories, roles, and credentials — version control is **non-negotiable**.

Introduction to Git Basics for Automation

Git is a **distributed version control system** that gives every team member a full copy of the repository. For automation teams, it becomes the single source of truth.



Basic Git Workflow



Key Commands

git clone	git add	git commit
git push	git branch	git checkout

Benefits of Storing Automation Code in Git

For AAP environments, Git isn't just a backup tool — it's a collaboration and governance platform. It lets you answer: *"Who changed the firewall rule automation last month?"* — and Git shows you exactly who, when, and what.

Version Tracking

Every change logged with author and timestamp

Rollback

Restore stable playbook in seconds

Peer Review

Pull requests enforce quality gates

CI/CD Integration

Trigger pipelines on code merge

Audit Trail

Full compliance history per commit

Multi-Env Branching

Dev, staging, and prod isolated by branch

Git Terminology Reference

These are the core Git concepts you'll encounter when connecting AAP to source control. In telecom and banking environments, **Git branches map directly to deployment environments**.

Term	Meaning
Repository	Storage location for all automation code and history
Commit	A snapshot of changes at a specific point in time
Branch	An independent line of development (e.g., dev, staging, prod)
Tag	A version marker used to label releases (v1.0, v2.0)
Merge	Combining two branches into one unified codebase
Pull Request	A structured code review and approval mechanism

4.2 Understanding Projects in AAP

AAP Projects are the bridge between the Controller and your automation code. This section explains how AAP pulls content, and why SCM-based projects are the only production-grade approach.

What Is a Project in AAP?

Definition

In Red Hat Ansible Automation Platform, a **Project** is a logical reference to automation content — playbooks, roles, and inventories.

> **Key insight:** AAP does NOT store playbooks in its database. It pulls them from external sources at runtime.

Supported Sources

- Local file system (manual)
- Git repositories (recommended)
- Other SCM systems

Project = Bridge between AAP and your automation code.

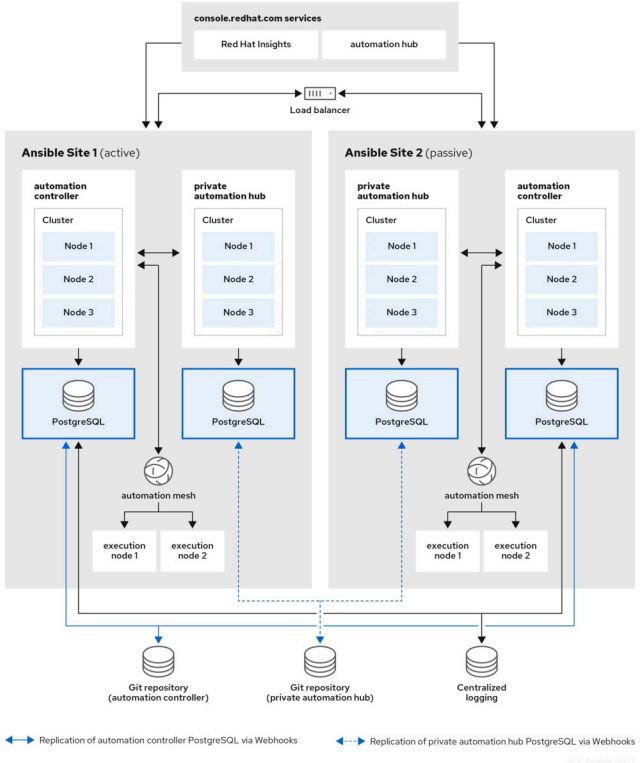
Manual Projects vs. SCM Projects

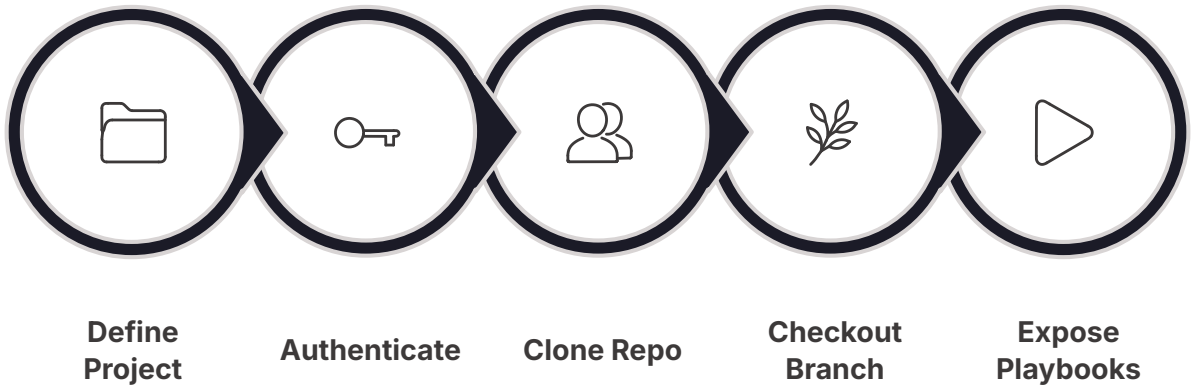
Manual Project	SCM Project (Git)
• Uses a local file system path • Content copied manually to server • No branch awareness • Not scalable • Not recommended for production	• Pulls content directly from Git • Supports branch selection • Supports auto-sync on push • Integrates with CI/CD pipelines • Enterprise-ready ✔️

Best Practice: Always use SCM-based projects in production environments. Manual projects are suitable for isolated lab exercises only.

How AAP Retrieves Automation Content

When a Project syncs, AAP performs a **Git clone/pull operation** behind the scenes. Understanding this flow helps troubleshoot sync failures and credential issues.





Project Sync = Git Pull operation executed inside the Automation Controller.

Project Synchronization Concepts

AAP supports four sync strategies. Choose based on your team's workflow maturity and compliance requirements.

1

Manual Sync

Triggered by an admin on demand. Suitable for controlled lab environments.

2

On Launch Sync

Syncs every time a Job Template runs. Ensures latest code is always used.

3

Scheduled Sync

Syncs at defined intervals. Ideal for regulated environments with change windows.

4

Webhook-Triggered Sync

Syncs automatically on Git push. Enables true GitOps-style automation.

Enterprise Tip: Always enable **"Update revision on launch"** in dynamic environments to guarantee job templates use the correct commit.

4.3 Working with Git Repositories

This section covers the hands-on mechanics: connecting AAP to GitHub or GitLab, creating projects from Git sources, managing branches per environment, and automating sync with webhooks.

Connecting AAP to GitHub / GitLab

Step-by-Step Setup

01

Create SCM Credential

In AAP, navigate to Credentials → Add. Select **Source Control** type.

02

Choose Auth Method

Use **HTTPS + Personal Access Token (PAT)** or SSH key pair.

03

Create a Project

Set SCM Type to **Git**, enter the repository URL, and attach the credential.

04

Sync & Verify

Click **Sync**. Confirm playbooks appear in Job Template dropdowns.

Authentication Options

HTTPS + PAT

Simpler setup, token-based, preferred for GitHub/GitLab

SSH Key Pair

More secure for server-to-server, no token expiry risk

Enterprise Recommendation: Use tokens or SSH keys — never plain passwords. Rotate tokens regularly and store them in a vault.

Creating Projects from Git Sources

When creating an SCM-backed Project in AAP, every field has an operational impact. Fill in all required fields before clicking **Sync**.

Field	Value / Notes	Impact
SCM Type	Git	Enables Git pull behavior
SCM URL	https://github.com/org/repo.git	Points to the source
Branch / Tag	main, dev, v1.2	Controls which code version is used
Credential	SCM credential with PAT or SSH	Authenticates the clone operation
Update Options	Update on launch, Clean	Governs sync behavior

After a successful sync, playbooks from the repository automatically populate the **Job Template Playbook dropdown**.

Understanding Branch Selection

In AAP, each Project can reference a specific Git branch, enabling you to isolate automation across environments using a simple branch strategy.

1

dev

Active development. Playbooks tested by engineers before promotion.

2

staging

Pre-production validation. Mirrors production configuration.

3

production

Live automation. Only merged, reviewed, approved code runs here.

Option A — Separate AAP Instances

One Controller per environment. Maximum isolation, higher infrastructure cost.

Option B — Single AAP, Multiple Projects

One Controller with separate Projects mapped to each branch. Cost-efficient, widely adopted.

Automatic vs. Manual Project Updates

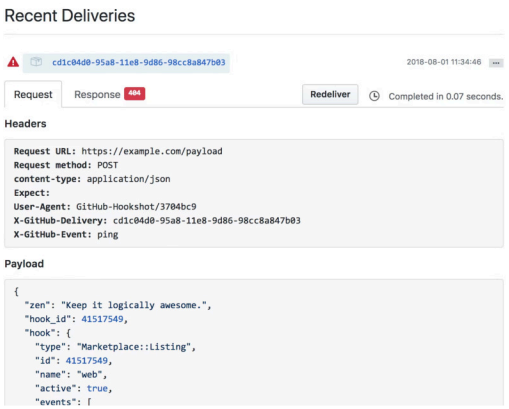
Match your sync strategy to your team's operational maturity and compliance requirements.

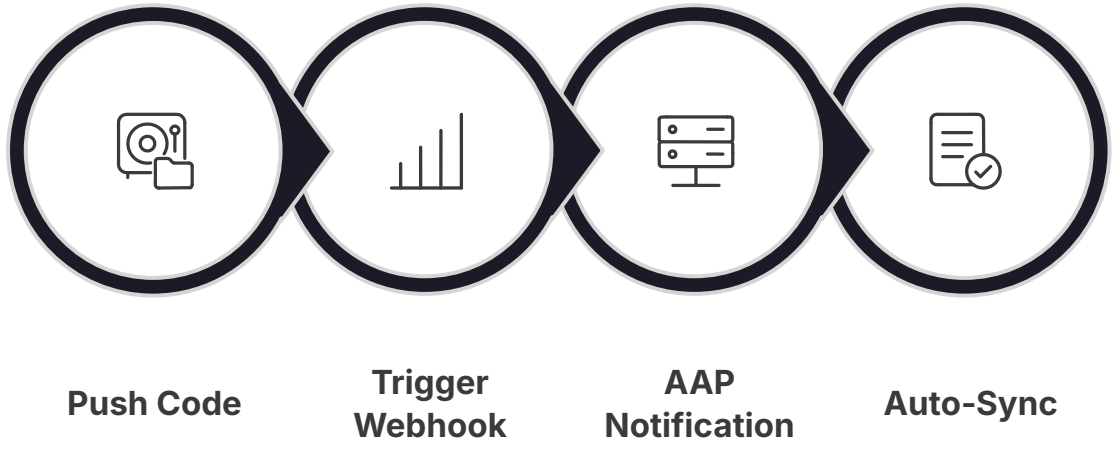
Sync Type	Best Use Case	Notes
Manual	Small labs, demos	Admin-triggered, full control
On Launch	Dynamic teams, fast iteration	Always runs latest code at job start
Scheduled	Regulated / change-window environments	Predictable sync intervals
Webhook	CI/CD driven pipelines	Push-triggered, fully automated

Enterprise Recommendation: Prefer **Webhook** or **Scheduled** updates in production. Manual sync introduces human delay and is error-prone at scale.

Setting Up Webhooks for Automatic Synchronization

Webhooks close the loop between your Git workflow and AAP — enabling a true **GitOps model** where a code push automatically triggers a project sync.





Configure the AAP Webhook URL in your GitHub/GitLab project settings under **Settings → Webhooks**. Use a secret token for security.

4.4 Organizing Automation Content

A well-structured repository is the foundation of maintainable, scalable, and compliant automation. This section defines the enterprise-recommended layout for AAP automation projects.

Recommended Directory Structure

Organizing your automation repository consistently improves reusability, maintenance, compliance, and scalability across teams and environments.

```
network-automation/
│
├── playbooks/
│   ├── configure_ntp.yml
│   ├── deploy_firewall.yml
│
├── roles/
│   ├── ntp/
│   ├── firewall/
│
├── inventories/
│   ├── dev/
│   ├── prod/
│
├── group_vars/
├── host_vars/
│
├── requirements.yml
└── README.md
```

Why This Structure Works

Reusability

Roles and variables are decoupled from playbooks and reusable across projects.

Maintainability

Clear separation of concerns makes updates isolated and predictable.

Compliance

Auditors can trace exactly what ran, where, and with which variables.

Scalability

Adding new playbooks, roles, or environments doesn't break existing structure.

Where to Place Roles and Variables

Consistent component placement prevents variable sprawl and keeps playbooks clean, readable, and portable across AAP environments.

Component	Location	Notes
Custom Roles	`roles/`	One directory per role; follows Ansible Galaxy structure
Group Variables	`group_vars/`	Applied to all hosts in an inventory group
Host Variables	`host_vars/`	Applied to a specific managed node
External Roles	`requirements.yml`	Pulled from Ansible Galaxy or Private Automation Hub

Best Practice: Never hardcode variables inside playbooks. Use `group_vars/` and `host_vars/` to keep playbooks environment-agnostic and reusable.

✔ Reusable

Playbooks work across dev, staging, and prod without modification

✔ Auditable

Variable changes are tracked in Git with full history

✔ Secure

Sensitive values stay in AAP credentials — never in repo files