

The Ultimate Guide to GitHub 2026

Intelligence, Collaboration & Code — Fully Declassified

EDITION

2026 · Q2

AUTHOR

MARIE-SOLEIL SESHAT LANDRY

CLASSIFICATION

OPEN SOURCE

PAGES

COMPREHENSIVE



GitHub in 2026 is no longer merely a version control platform — it is the intelligence layer of the global software ecosystem. From AI-driven pair programming to enterprise-grade security orchestration, GitHub has evolved into the command centre where code, collaboration, and automation converge. This guide decodes every critical feature, workflow, and emerging practice you need to operate at the highest level.

◆ Table of Contents

FOUNDATIONS

- [01 · GitHub Platform Overview 2026](#)
(#ch1)
- [02 · Repositories — The Deep Dive](#)
(#ch2)
- [03 · Branching & Git Workflows](#) (#ch3)
- [04 · Pull Requests & Code Review](#) (#ch4)

AI & AUTOMATION

- [05 · GitHub Copilot 2026](#) (#ch5)
- [06 · GitHub Actions — Full Reference](#)
(#ch6)

SECURITY & OPERATIONS

- [08 · Security & Dependabot](#) (#ch8)
- [09 · GitHub Packages & Container Registry](#) (#ch9)
- [10 · GitHub Projects & Planning](#) (#ch10)

ADVANCED & REFERENCE

- [11 · GitHub CLI & API Mastery](#) (#ch11)
- [12 · Enterprise & Org Management](#)
(#ch12)
- [13 · Codespaces & Dev Environments](#)
(#ch13)

CHAPTER 01

GitHub Platform Overview 2026

The state of the platform, what changed, and what it means for you.

GitHub has passed **200 million repositories** and serves over **100 million developers** globally as of 2026. The Microsoft-owned platform has undergone a profound transformation — it is now best understood as a **developer intelligence platform** that unifies code hosting, AI assistance, security scanning, package management, CI/CD, and project planning under a single unified interface.

What's New in 2026



Copilot Workspace

End-to-end AI task execution — from issue triage to code, test, and PR creation without leaving GitHub.com.



GitHub Models GA

Access to frontier AI models (GPT-4o, Claude, Llama) directly from your repo for prototyping and deployment.



Security Campaigns

Orchestrate bulk remediation of CVEs across thousands of repos with coordinated developer workflows.



Actions GPU Runners

Native GPU-accelerated runners for ML training, inference, and scientific workloads in CI/CD.



Artifact Attestations

Cryptographic provenance for every build artifact — SLSA Level 3 compliance built into Actions.



Merge Queues (GA)

Enterprise-grade merge orchestration eliminating the "merge race" on high-velocity main branches.

GitHub's 2026 trajectory positions AI not as a peripheral add-on but as the execution layer of software development. The strategic implication: developers who master AI-augmented GitHub workflows will achieve 3–5× higher throughput on complex engineering tasks.

GitHub Plans at a Glance

PLAN	TARGET	KEY LIMITS	COPILOT	ACTIONS MINUTES
Free	Individuals & OSS	2,000 CI minutes/mo, 500 MB packages	2,000 completions/mo FREE	2,000
Pro	Individual devs	3,000 CI minutes/mo, 2 GB packages	Unlimited PRO	3,000
Team	Small teams	3,000 CI minutes/mo, CODEOWNERS	Unlimited PRO	3,000
Enterprise	Orgs & enterprises	50,000 CI minutes/mo, SSO, SAML	Enterprise NEW	50,000+

CHAPTER 02

Repositories — The Deep Dive

Architecture, settings, and professional-grade repo management.

A GitHub repository is the atomic unit of software intelligence — it stores not just code but history, intent, context, and collaboration. Understanding repository anatomy at depth separates casual users from platform professionals.

Creating Repositories with Intention

Every repository decision at creation time has downstream consequences. Apply these principles:

- ◆ **Naming conventions:** Use lowercase kebab-case (`my-project-name`). Avoid underscores for public repos — URLs are cleaner.

- ◆ **Visibility:** Public repositories contribute to your OSINT footprint. Private repos consume storage quota. Internal (Enterprise only) repos are visible org-wide but not publicly indexed.
- ◆ **Initialize with a README:** This creates the default branch immediately, avoiding the empty-repo UX trap. Include project purpose, status, and license in the first commit.
- ◆ **.gitignore:** Always select the language-appropriate template at creation. Retroactively adding one does not remove already-tracked files.
- ◆ **License selection:** MIT for maximum reuse, Apache 2.0 for patent protection, GPL for copyleft, AGPL for SaaS copyleft. Unlicensed = all rights reserved by default.

```
# Create a repo and push initial code via GitHub CLI
gh repo create my-intelligence-project \
  --public \
  --license mit \
  --gitignore Python \
  --description "OSINT toolchain for open-source analysis"

gitclone https://github.com/username/my-intelligence-project
cd my-intelligence-project
# Start building immediately - .gitignore and LICENSE already present
```

Repository Settings That Matter

⚠ OPERATIONAL WARNING

Repository settings are not audited by default on the Free/Pro plans. On Enterprise, enable **audit log streaming** to capture all settings changes. A single misconfigured branch protection rule can allow force-pushes to main and permanently rewrite history.

BRANCH PROTECTION RULES (2026)

- ◆ **Require pull request reviews:** Set minimum 1–2 approvals; enable "Dismiss stale reviews on new commits."
- ◆ **Require status checks:** Lock required CI checks that must pass before merge — prevents merging broken code.
- ◆ **Require signed commits:** GPG or SSH signing ensures author authenticity — critical for supply chain integrity.
- ◆ **Restrict force pushes:** Never allow force-push to `main` in production codebases.
- ◆ **Require linear history:** Enforces squash or rebase merges — keeps git history readable.

RULESETS (NEW ARCHITECTURE)

GitHub Rulesets replaced the older branch protection model with a more powerful, layered system. Rulesets can target multiple branches via patterns, support bypass lists, and apply organization-wide. They are the recommended approach for all new configurations in 2026.

```
# List rulesets on a repo via CLI
gh api repos/{owner}/{repo}/rulesets \
  --jq '.[ ] | {id:.id, name:.name, enforcement:.enforcement}'
```

Repository Best Practices

PRACTICE	WHY IT MATTERS	TOOL
CODEOWNERS file	Automatic review assignment by file path	<code>.github/CODEOWNERS</code>
Issue templates	Structured bug reports & feature requests	<code>.github/ISSUE_TEMPLATE/</code>
PR templates	Consistent PR descriptions; checklist enforcement	<code>.github/pull_request_template.md</code>
Dependabot config	Automated dependency updates	<code>.github/dependabot.yml</code>
Security policy	Responsible disclosure contact for vuln reporters	<code>SECURITY.md</code>
Contributing guide	Onboards external contributors efficiently	<code>CONTRIBUTING.md</code>

CHAPTER 03

Branching & Git Workflows

Proven strategies for managing parallel development without chaos.

Branching strategy is the DNA of a team's development velocity. The wrong model creates bottlenecks and merge nightmares; the right model creates a frictionless flow from idea to production.

The Four Major Branching Strategies



GitHub Flow

One main branch + short-lived feature branches. Simple, fast, ideal for continuous deployment. The dominant model in 2026 for cloud-native teams.



Git Flow

main + develop + feature/release/hotfix branches. Structured but complex. Best for versioned, scheduled release software.



Trunk-Based Dev

All devs commit directly to trunk (main) with feature flags. Maximum velocity, requires strong CI/CD and testing discipline.



Forking Workflow

Contributors fork the repo, work in their fork, then PR to upstream. The standard model for open-source projects.

Recommended: GitHub Flow (2026 Edition)

1

Create a branch from main

Name it descriptively: `feat/copilot-integration`, `fix/auth-null-check`, `docs/api-v3-update`. Branch names are OSINT — they reveal your work.

2

Make your changes & commit often

Each commit should represent one logical unit of work. Use conventional commits: `feat:`, `fix:`, `docs:`, `chore:`. This powers automatic changelogs.

3

Open a Pull Request early

Open a

Draft PR

as soon as you push. This triggers CI, notifies reviewers, and makes your work visible. Switch to "Ready for Review" when done.

4

Review, discuss, iterate

Address comments as commits. Use PR threads to track every decision. GitHub Copilot Code Review now auto-comments on potential issues.

5

Merge via Merge Queue

For high-velocity teams, use the Merge Queue. It runs CI on the queued-merge version of the code before the merge completes, eliminating broken-main incidents.

```
# Complete GitHub Flow in commands
git switch -c feat/intelligence-module
# ... write code ...
git add -p                                # stage interactively – review your own di
git commit -m "feat: add OSINT data ingestion module"
git push -u origin feat/intelligence-module
gh pr create --draft --title "feat: OSINT data ingestion" \
  --body "Closes #42. Adds structured OSINT feed processing."
# ... CI runs, reviews happen ...
gh pr merge --squash --delete-branch
```

CHAPTER 04

Pull Requests & Code Review

The art and science of collaborative code improvement.

Pull Requests are the highest-leverage collaboration surface on GitHub. A well-crafted PR reduces review time, reduces bugs, and serves as permanent institutional knowledge. A poorly crafted PR wastes everyone's time and introduces hidden risk.

Anatomy of an Elite Pull Request

- ◆ **Title:** Conventional commit format. `feat(auth): add OAuth2 PKCE flow` — reviewers know the scope before opening.
- ◆ **Description:** What changed, why it changed, how to test it, what was explicitly NOT changed. Link to the issue with `Closes #N`.
- ◆ **Screenshots / recordings:** For UI changes, always include before/after visuals.
- ◆ **Checklist:** Verify your PR template items — tests added, docs updated, migrations handled.
- ◆ **Size:** PRs under 400 lines of change have dramatically higher review quality. Break large PRs into a stack.

Copilot Code Review (2026)

GitHub Copilot now functions as an autonomous code reviewer. When enabled, it automatically reviews every PR and posts inline comments on:

- ◆ Security anti-patterns (hardcoded secrets, SQL injection vectors, XSS risks)
- ◆ Performance regressions (N+1 queries, unbounded loops)

- ◆ Code style inconsistencies relative to the codebase
- ◆ Missing error handling and null checks
- ◆ Test coverage gaps

PRO TECHNIQUE

Enable **Copilot as a required reviewer** in branch protection rules. This ensures AI review runs on every PR before human review begins — humans can then focus on architecture and intent rather than catching syntax issues.

Stacked Pull Requests

Stacking is the practice of building a series of PRs where each one builds on the previous — enabling parallel review of large, interdependent features. GitHub natively displays stacked PR relationships in 2026 via the "PR stack" sidebar.

```
# Create a stack: feature requires DB migration + API layer + UI
git switch -c feat/db-schema main
# ... write migration ...
gh pr create --base main --title "feat: add user_sessions table"

git switch -c feat/api-sessions feat/db-schema
# ... write API ...
gh pr create --base feat/db-schema --title "feat: session management API"

git switch -c feat/ui-sessions feat/api-sessions
# ... write UI ...
gh pr create --base feat/api-sessions --title "feat: session management UI"
```

CHAPTER 05

GitHub Copilot 2026

The AI co-pilot that has redefined software development velocity.

GitHub Copilot in 2026 is not a code autocomplete tool. It is an autonomous software development agent that can plan, research, write, test, and deploy — under your direction.

Copilot Tiers & Capabilities

TIER	KEY FEATURES	CONTEXT WINDOW	AGENTS
Copilot Free	2,000 completions/mo, 50 chat messages/mo	8K tokens	✗
Copilot Pro	Unlimited completions, chat, code review	64K tokens	Limited
Copilot Pro+	Premium models, Copilot Workspace, extended agent runs	128K tokens	✓ Full
Copilot Enterprise	Org knowledge base, codebase indexing, policy controls	256K tokens	✓ Full

Copilot Workspace

Copilot Workspace is the 2026 flagship feature — a fully AI-driven software development environment embedded in GitHub.com. The workflow:

1

Start from an issue

Open any GitHub issue and click "Open in Copilot Workspace." The AI reads the issue, the codebase, related issues, and relevant PRs.

2

AI generates a plan

Copilot produces a multi-step implementation plan. You review, edit, and approve each step before execution.

3

Code is generated & tested

Copilot writes the implementation across multiple files, runs tests, identifies failures, and self-corrects in a loop.

4

PR is opened automatically

Once passing, Copilot opens a PR with full description, test results, and links to the original issue — ready for your review.

Effective Copilot Prompting

Copilot quality scales directly with prompt quality. Apply these intelligence-grade prompting principles:

- ◆ **Provide context first:** Tell Copilot the language, framework, coding style, and constraints before asking for code.
- ◆ **Be specific about what NOT to do:** "Do not use global state. Do not mutate the input array. Use `async/await`, not callbacks."
- ◆ **Ask for tests first:** "Write a failing test for [behavior], then write the code to make it pass." This is test-driven prompting.
- ◆ **Iterate with feedback:** Copilot Chat remembers the conversation. Build on each response rather than starting fresh.
- ◆ **Use `/` slash commands:** `/explain`, `/fix`, `/tests`, `/doc` target specific sub-tasks precisely.

SPYMASTER'S INSIGHT

The most productive Copilot users in 2026 treat the AI as a **junior analyst**: highly capable at executing well-defined tasks, but requiring clear direction, explicit constraints, and senior review. The bottleneck shifts from writing code to *specifying intent* — a skill that rewards analytical clarity above raw typing speed.

CHAPTER 06

GitHub Actions — Full Reference

The automation engine that powers modern software delivery.

GitHub Actions is a **workflow automation platform** deeply integrated into GitHub events. In 2026, it powers CI/CD, security scanning, release management, repository maintenance, data pipelines, and ML model training — all from YAML workflow files in your repository.

Core Concepts

- ◆ **Workflow:** A YAML file in `.github/workflows/` that defines automation. Triggered by events.
- ◆ **Event:** The trigger — `push`, `pull_request`, `schedule`, `workflow_dispatch`, `release`, and 40+ others.
- ◆ **Job:** A set of steps that run on the same runner. Jobs run in parallel by default.
- ◆ **Step:** An individual task — either a shell command or a reusable **Action**.

- ◆ **Action:** A reusable unit from the GitHub Marketplace or your own repository.
- ◆ **Runner:** The compute environment — GitHub-hosted (Ubuntu, Windows, macOS, GPU) or self-hosted.

Production-Grade CI/CD Workflow

```
# .github/workflows/ci-cd.yml – Production CI/CD Pipeline
name: Production CI/CD

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${ github.repository }

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.11", "3.12"]
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: ${ matrix.python-version }
          cache: 'pip'
      - run: pip install -r requirements.txt
      - run: pytest --cov=src --cov-report=xml
      - uses: codecov/codecov-action@v4

  security-scan:
    runs-on: ubuntu-latest
    permissions:
      security-events: write
    steps:
      - uses: actions/checkout@v4
      - name: Run CodeQL Analysis
        uses: github/codeql-action/analyze@v3
        with:
          languages: python

  build-and-push:
    needs: [test, security-scan]
    runs-on: ubuntu-latest
    if: github.event_name == 'push'
    permissions:
      contents: read
      packages: write
```

```

    id-token: write    # for artifact attestation
  steps:
    - uses: actions/checkout@v4
    - uses: docker/setup-buildx-action@v3
    - uses: docker/login-action@v3
      with:
        registry: ${ env.REGISTRY }
        username: ${ github.actor }
        password: ${ secrets.GITHUB_TOKEN }
    - uses: docker/build-push-action@v5
      id: push
      with:
        push: true
        tags: ghcr.io/${ github.repository }:latest
        cache-from: type=gha
        cache-to: type=gha,mode=max
    - name: Generate Artifact Attestation # SLSA provenance
      uses: actions/attest-build-provenance@v1
      with:
        subject-name: ghcr.io/${ github.repository }
        subject-digest: ${ steps.push.outputs.digest }

```

Actions Security Best Practices

SECURITY ALERT

Pin all third-party Actions to a full commit SHA, not a tag. Tags are mutable —

`actions/checkout@v4` can be changed by the author without notice. Use

`actions/checkout@11bd71901bbe5b1630ceea73d27597364c9af683` instead. This prevents supply chain attacks via compromised Action tags.

- ◆ Use **GITHUB_TOKEN** over PATs wherever possible — it's scoped to the repo and expires.
- ◆ Set **permissions at the job level**, not workflow level, using principle of least privilege.
- ◆ Never print secrets in logs — even accidentally with `echo $SECRET_VAR`.
- ◆ Use **environment protection rules** to require manual approval before deploying to production.
- ◆ Enable **Actions audit logging** and alert on unexpected workflow runs from forks.

CHAPTER 07

GitHub Models & AI Agents

Frontier AI models, directly accessible from your repositories.

GitHub Models (GA since 2025, expanded in 2026) provides direct access to frontier large language models from within GitHub — enabling rapid prototyping, evaluation, and deployment of AI-powered applications without leaving the platform.

Available Models (2026)

MODEL	PROVIDER	STRENGTHS	CONTEXT
GPT-4o	OpenAI	General reasoning, multimodal, code	128K
Claude Sonnet 4	Anthropic	Analysis, writing, long-context reasoning	200K
Llama 3.3 70B	Meta	Open weights, cost-efficient	128K
Mistral Large	Mistral AI	European compliance, multilingual	128K
Phi-4	Microsoft	Small, fast, on-device capable	16K
DeepSeek-R1	DeepSeek	Mathematical reasoning, STEM	64K

Using GitHub Models via API

```
import os
from openai import OpenAI

# GitHub Models uses the OpenAI SDK with a different base URL
client = OpenAI(
    base_url="https://models.inference.ai.azure.com",
    api_key=os.environ["GITHUB_TOKEN"]
)

response = client.chat.completions.create(
    model="claude-sonnet-4-20251018",
    messages=[
        {"role": "system", "content": "You are an expert OSINT analyst."},
        {"role": "user", "content": "Summarize the key risks in this dataset."}
    ],
    max_tokens=2048
)

print(response.choices[0].message.content)
```

INTELLIGENCE NOTE

GitHub Models rates are significantly lower than direct API calls during the free tier. For prototyping and evaluation, use GitHub Models before committing to a provider's paid plan. Rate limits apply per user, not per repository.

CHAPTER 08

Security & Dependabot

Defending your codebase — automated intelligence on vulnerabilities.

GitHub's security suite is now the most comprehensive free security platform available to developers. In 2026, it covers dependency scanning, secret detection, static analysis, container scanning, and AI-driven remediation — all natively integrated.

The GitHub Security Stack

- ◆ **Dependabot Alerts:** Detects known CVEs in your dependencies by cross-referencing the GitHub Advisory Database (GHSA). Free for all repositories.
- ◆ **Dependabot Security Updates:** Automatically opens PRs to update vulnerable dependencies to patched versions.
- ◆ **Dependabot Version Updates:** Keeps all dependencies current, not just vulnerable ones. Configured via `dependabot.yml`.
- ◆ **Secret Scanning:** Detects 200+ secret patterns (API keys, tokens, credentials) committed to code. Free for public repos; Advanced Security for private.
- ◆ **Push Protection:** Blocks commits containing detected secrets *before* they reach the remote — the critical prevention layer.
- ◆ **Code Scanning (CodeQL):** SAST analysis identifying security vulnerabilities and quality issues. Supports 10+ languages.
- ◆ **Security Campaigns:** New in 2026 — orchestrate cross-repo remediation of shared vulnerabilities at the org level.

```
# .github/dependabot.yml – Comprehensive Dependabot configuration
version: 2
updates:
  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
      day: "monday"
    open-pull-requests-limit: 10
    groups:
      dev-dependencies:
        patterns: ["pytest*", "black", "mypy"]
        reviewers: ["security-team"]
        labels: ["dependencies", "security"]

  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      interval: "weekly"

  - package-ecosystem: "docker"
    directory: "/"
    schedule:
      interval: "daily"      # containers need rapid patching
```

CRITICAL SECURITY PRACTICE

Enable Push Protection on every repository without exception. Secret scanning after the fact is remediation. Push protection is prevention. Once a secret is pushed — even briefly — it must be considered compromised and rotated immediately. This is non-negotiable operational security.

CHAPTER 09

GitHub Packages & Container Registry

Publishing, distributing, and consuming software packages at scale.

GitHub Packages is an **artifact registry** co-located with your source code. It supports Docker/OCI containers, npm, Maven, NuGet, RubyGems, and generic artifact storage — all authenticated with the same `GITHUB_TOKEN` used in Actions.

Container Registry (ghcr.io)

The GitHub Container Registry at ghcr.io is the recommended container registry for GitHub-hosted projects. Key advantages:

- ◆ Free storage for public images (no rate limits for anonymous pulls)
- ◆ Fine-grained access control inherited from repository or organization permissions
- ◆ Artifact attestations for SLSA provenance built into Actions workflows
- ◆ Multi-architecture image support (amd64, arm64, armv7)
- ◆ OCI-compliant — works with any container tooling

```
# Authenticate, build, and push to ghcr.io
echo $GITHUB_TOKEN | docker login ghcr.io -u USERNAME --password-stdin

docker buildx build \
  --platform linux/amd64,linux/arm64 \
  --tag ghcr.io/username/my-app:1.0.0 \
  --tag ghcr.io/username/my-app:latest \
  --push .

# Verify the attestation
gh attestation verify oci://ghcr.io/username/my-app:1.0.0 \
  --owner username
```

CHAPTER 10

GitHub Projects & Planning

The native project management layer, deeply integrated with your code.

GitHub Projects v2 is a powerful, spreadsheet-like project management system with deep repository integration. It tracks issues and PRs across multiple repositories with custom fields, views, and automation — replacing the need for separate tools in many cases.

Core Capabilities



Multiple Views

Switch between Board (kanban), Table (spreadsheet), Roadmap (timeline), and custom layouts per project.



Custom Fields

Add text, number, date, single-select, iteration, and milestone fields to any item. Track priority, team, story points, and more.



Built-in Automation

Auto-add issues by label, auto-archive closed items, auto-set status on PR events — without writing a single Action workflow.



Insights Charts

Generate burn-up, burn-down, and throughput charts directly from project data — native velocity tracking.

Workflow Automation with GraphQL

```
# Add an issue to a project via GitHub CLI + GraphQL
gh project item-add 1 \
  --owner username \
  --url https://github.com/username/repo/issues/42

# Update a custom field value on a project item
gh api graphql -f query='
mutation {
  updateProjectV2ItemFieldValue(input: {
    projectId: "PVT_kwXXXXX"
    itemId: "PVTI_lAXXX"
    fieldId: "PVTF_lAXXX"
    value: { singleSelectOptionId: "OPTION_ID" }
  }) {
    projectV2Item { id }
  }
}'
```

Operate GitHub at the speed of thought — from the terminal.

The **GitHub CLI** (`gh`) is the professional operator's primary interface to GitHub. Installed on any system, it authenticates once and provides immediate command-line access to every GitHub feature — repositories, issues, PRs, Actions, releases, and the REST and GraphQL APIs.

Essential CLI Commands

REPOSITORY OPERATIONS

<code>gh repo create NAME</code>	Create a new repository interactively or with flags
<code>gh repo clone OWNER/REPO</code>	Clone a repository and set upstream remote
<code>gh repo view --web</code>	Open current repo in browser
<code>gh repo fork --clone</code>	Fork and clone in one command

PULL REQUEST OPERATIONS

<code>gh pr create --draft</code>	Create a draft pull request
<code>gh pr list --state open</code>	List all open pull requests
<code>gh pr checkout 42</code>	Check out PR #42 locally
<code>gh pr review --approve</code>	Approve the current PR
<code>gh pr merge --squash --delete-branch</code>	Squash merge and delete branch

ACTIONS & WORKFLOW OPERATIONS

<code>gh workflow list</code>	List all workflows in the repository
<code>gh workflow run WORKFLOW</code>	Manually trigger a workflow (workflow_dispatch)
<code>gh run list --limit 10</code>	List the last 10 workflow runs
<code>gh run watch RUN_ID</code>	Watch a running workflow in real-time
<code>gh run download RUN_ID</code>	Download artifacts from a completed run

ADVANCED API & SCRIPTING

<code>gh api repos/{owner}/{repo}</code>	Make a REST API call with auto-auth
<code>gh api graphql -f query='...'</code>	Execute a GraphQL query

<code>gh api --paginate /repos/{o}/{r}/issues</code>	Paginate through all results automatically
<code>gh secret set SECRET_NAME</code>	Set a repository or org secret
<code>gh extension install OWNER/EXT</code>	Install a CLI extension from GitHub

Scripting GitHub with the API

```
#!/bin/bash
# Export all open issues from a repo to JSON
gh api --paginate \
  -H "Accept: application/vnd.github+json" \
  /repos/OWNER/REPO/issues \
  --jq '[.[] | select(.state=="open") | {
    number: .number,
    title: .title,
    labels: [.labels[].name],
    created: .created_at,
    assignee: .assignee?.login
  }]' | jq -s '.' > issues-export.json

echo "Exported $(cat issues-export.json | jq length) issues"
```

CHAPTER 12

Enterprise & Org Management

Governing GitHub at scale — policies, compliance, and organizational architecture.

GitHub Enterprise Cloud provides the governance layer required by organizations operating at scale, with regulatory obligations, or with complex team structures. Understanding org architecture is prerequisite for effective large-scale deployment.

Organization Architecture

- ◆ **Enterprise Account:** Top-level billing and policy entity. Contains multiple Organizations. Enterprise owners set policies that override org settings.
- ◆ **Organization:** The primary unit of team and repo organization. Has owners, members, and teams. Billing rolls up to the Enterprise.
- ◆ **Teams:** Hierarchical groups within an org. Teams are assigned to repositories with Read, Triage, Write, Maintain, or Admin permissions.

- ◆ **Repositories:** Owned by orgs or individuals. Visibility: public, private, or internal (org-wide).

Enterprise Policy Controls (2026)

POLICY AREA	CONTROL	ENFORCEMENT LEVEL
Authentication	Require SAML SSO, enforce 2FA	Enterprise
IP Allow Lists	Restrict access by IP CIDR range	Enterprise / Org
Actions Permissions	Allow only internal / selected Actions	Enterprise / Org
Copilot Policies	Enable/disable features, content exclusions	Enterprise / Org
Audit Log Streaming	Stream to Splunk, Datadog, S3, etc.	Enterprise
Repository Insights	Org-wide contribution and activity dashboards	Org
Managed Users (EMU)	Identity managed by IdP — no personal accounts	Enterprise

▲ COMPLIANCE NOTE

Organizations subject to SOC 2, ISO 27001, or FedRAMP should enable **audit log streaming** immediately on provisioning an Enterprise account. The default 90-day audit log retention is insufficient for most compliance frameworks. Stream to an external SIEM for long-term retention and correlation.

CHAPTER 13

Codespaces & Dev Environments

Zero-setup cloud development environments — instant, consistent, disposable.

GitHub Codespaces provisions a full VS Code development environment in the cloud within 30–60 seconds. It is configured entirely via the `.devcontainer/` directory in your repository, ensuring every contributor has an identical, reproducible environment.

devcontainer.json Reference

```
{
  "name": "Intelligence Platform Dev",
  "image": "mcr.microsoft.com/devcontainers/python:3.12",

  "features": {
    "ghcr.io/devcontainers/features/github-cli:1": {},
    "ghcr.io/devcontainers/features/docker-in-docker:2": {},
    "ghcr.io/devcontainers/features/node:1": {"version": "20"}
  },

  "postCreateCommand": "pip install -r requirements.txt && pre-commit install",

  "customizations": {
    "vscode": {
      "extensions": [
        "GitHub.copilot",
        "ms-python.python",
        "ms-python.mypy-type-checker",
        "charliermarsh.ruff"
      ],
      "settings": {
        "python.defaultInterpreterPath": "/usr/local/bin/python",
        "editor.formatOnSave": true
      }
    }
  },

  "forwardPorts": [8000, 5432, 6379],
  "portsAttributes": {
    "8000": {"label": "Dev Server", "onAutoForward": "openPreview"}
  }
}
```

Codespaces Best Practices

- ◆ **Prebuild codespaces** for large projects: Prebuilds cache the post-create state, reducing startup time from 3+ minutes to under 30 seconds.
- ◆ **Use Codespaces secrets** for API keys — they're injected as environment variables, never committed to repo.
- ◆ **Default to the smallest machine type** (2-core) — upgrade only when actually needed. CPU is rarely the bottleneck.
- ◆ **Stop codespaces when not in use.** Idle codespaces consume your monthly quota. The auto-stop timeout defaults to 30 minutes.

Command Reference & Cheatsheet

The professional's quick-reference — print and keep at your station.

Essential Git Commands

DAILY GIT OPERATIONS	
<code>git switch -c BRANCH</code>	Create and switch to a new branch (modern syntax)
<code>git add -p</code>	Stage changes interactively (hunk-by-hunk)
<code>git commit --amend --no-edit</code>	Add staged changes to last commit without editing message
<code>git stash push -m "description"</code>	Stash with a meaningful name
<code>git log --oneline --graph --all</code>	Visual branch graph in terminal
<code>git diff HEAD~1</code>	Compare working tree to one commit ago
<code>git bisect start</code>	Begin binary search for the commit that introduced a bug
<code>git reflog</code>	Show history of HEAD movements — rescue lost commits
<code>git cherry-pick SHA</code>	Apply a specific commit to the current branch
<code>git rebase -i HEAD~N</code>	Interactive rebase to squash, reorder, or edit N commits

REMOTE & SYNC OPERATIONS	
<code>git fetch --prune</code>	Fetch and remove deleted remote branches locally
<code>git pull --rebase</code>	Rebase instead of merge on pull — cleaner history
<code>git push --force-with-lease</code>	Safe force-push — fails if remote has new commits
<code>git remote add upstream URL</code>	Add original repo as upstream (for forks)
<code>git fetch upstream && git rebase upstream/main</code>	Sync fork with upstream

Conventional Commits Reference

TYPE	WHEN TO USE	EXAMPLE
<code>feat</code>	New feature for the user	<code>feat(auth): add biometric login</code>
<code>fix</code>	Bug fix for the user	<code>fix(api): resolve null pointer in user lookup</code>
<code>docs</code>	Documentation only changes	<code>docs: update API authentication guide</code>
<code>style</code>	Formatting, no logic change	<code>style: apply black formatting</code>
<code>refactor</code>	Code restructure, no feature/fix	<code>refactor(db): extract query builder class</code>
<code>test</code>	Adding or correcting tests	<code>test: add unit tests for session manager</code>
<code>chore</code>	Build, dependency, tooling	<code>chore: upgrade pytest to 8.x</code>
<code>perf</code>	Performance improvement	<code>perf(cache): implement Redis query caching</code>
<code>ci</code>	CI/CD configuration changes	<code>ci: add GPU runner for ML tests</code>
<code>revert</code>	Revert a prior commit	<code>revert: feat(auth): add biometric login</code>

GitHub Keyboard Shortcuts

SHORTCUT	ACTION	CONTEXT
<code>.</code> (period)	Open repo in github.dev (browser VS Code)	Any repo
<code>Ctrl+K</code>	Open command palette / search	Anywhere
<code>T</code>	Activate file search in repo	Repo view
<code>L</code>	Jump to a line in file viewer	File view
<code>B</code>	Open git blame for current file	File view
<code>W</code>	Switch branch/tag	Repo view
<code>G N</code>	Go to Notifications	Anywhere

SHORTCUT	ACTION	CONTEXT
Ctrl+Enter	Submit comment	Comment box
R	Quote reply to comment	Issue / PR

 **FINAL INTELLIGENCE SUMMARY**

GitHub in 2026 rewards those who invest in **depth over breadth**. Master your workflow, automate your repetitive tasks with Actions, leverage Copilot as a force multiplier rather than a replacement, and treat security as a default rather than an afterthought. The developers who understand the full stack of GitHub's capabilities — from git internals to AI agents — hold a decisive operational advantage.

The mission never ends. The code always ships.



 **About This Document**

This guide is published by **Marie Landry's Spy Shop** (marielandryspyshop.com (<https://marielandryspyshop.com>)), an ethics-first OSINT and intelligence services practice operated by Marie-Soleil Seshat Landry, CEO & Founder of Landry Industries Inc., Moncton, New Brunswick, Canada. ORCID: [0009-0008-5027-3337](https://orcid.org/0009-0008-5027-3337) (<https://orcid.org/0009-0008-5027-3337>). Content reflects platform status as of April 2026. GitHub features evolve rapidly — verify current behaviour at docs.github.com (<https://docs.github.com>).