DUMPSBOSS.

GitHub Foundations

Microsoft GH-900

Version Demo

Total Demo Questions: 17

Total Premium Questions: 179

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Describe the fundamentals of GitHub	45
Topic 2, Describe the GitHub flow	8
Topic 3, Describe collaboration features of GitHub	88
Topic 4, Describe modern development workflows	26
Topic 5, Describe GitHub security	12
Total	179

QUESTION NO: 1

Which two statements correctly describe how GitHub Actions secrets can be used in a deployment workflow?

(Each answer presents a complete solution. Choose two.)

A. Repository secrets can be made available to workflows in that repository.

B. Environment secrets can be made available to jobs that reference the environment.

C. Secrets should be committed to the workflow file so that runs are reproducible.

D. Public repository variables provide the same protection as secrets.

E. Branch protection rules store encrypted workflow secrets.

ANSWER: A B

Explanation:

Repository secrets can be made available to workflows in that repository. GitHub Actions supports secrets scoped to a repository, enabling workflows in that repository to retrieve sensitive values through the `secrets` context. This is appropriate for credentials, tokens, and configuration values that several workflows may need while keeping their literal values out of workflow YAML and logs. GitHub encrypts secrets at rest and redacts registered secret values from workflow logs where possible.

Environment secrets can be made available to jobs that reference the environment. Environments provide deployment-focused scoping for secrets. For example, a job that deploys to production can declare the production environment and then access secrets configured specifically for that environment. This supports separation of staging and production credentials and works with environment protection rules, such as required reviewers or deployment branch restrictions, before a job can access environment-scoped secrets. These controls help ensure that sensitive deployment credentials are supplied only in the intended deployment context. See Using secrets in GitHub Actions and Using environments for deployment.

QUESTION NO: 2

What is the relationship between commits and pull requests on GitHub?

A. Commits are made on a branch that can have a linked pull request.

B. Commits can only be made after a pull request is created.

C. Commits can only be made before a pull request is created.

D. Commits are made on a pull request that can have a linked branch.

ANSWER: A

Explanation:

Commits are made on a branch that can have a linked pull request. A Git commit records a snapshot of changes in a repository's history, and it extends the history of the branch currently checked out when the commit is created. A pull request is then opened to propose merging changes from a source (head) branch into a destination (base) branch. It provides a place to review, discuss, approve, and merge the work represented by the branch's commits and file changes.

The relationship remains dynamic while the pull request is open. After a developer opens a pull request for a feature branch, any later commits pushed to that same head branch are automatically included in the pull request. This allows reviewers to inspect the updated comparison and commit history without requiring a separate pull request for each additional commit. In practice, teams commonly commit work locally, push the branch to GitHub, and open a pull request when the branch is ready for collaboration or review. GitHub describes pull requests as proposals to merge changes between branches and notes that commits pushed to the head branch are included in the pull request. See About pull requests and About commits.

QUESTION NO: 3

An organization has a starter repository that provides a standard directory structure, README file, and workflow files. Teams must create new projects from this structure without inheriting the starter repository's commit history. Which GitHub feature should the organization use?

A. A template repository

B. A fork

C. A clone

D. A branch

ANSWER: A

Explanation:

A template repository is the appropriate feature because it enables an organization to create a reusable project starting point that other teams can use to generate new repositories. When a team creates a repository from a template, GitHub copies the template repository's current files, directory structure, and relevant repository configuration into a newly created, independent repository. This directly supports consistent project scaffolding, including standard documentation and workflow files, while allowing each project to evolve separately.

Crucially, repositories created from templates start without the template repository's commit history. The generated repository has its own initial commit and its own independent history, so teams can manage changes, branches, releases, and pull requests for their project without carrying the starter repository's prior development record. This makes templates well suited for organization-approved boilerplate and repeatable repository layouts. GitHub documents that template repositories let users create new repositories with the same files and folders as the template, but without its commit history. See Creating a template repository and Creating a repository from a template.

QUESTION NO: 4

A pull request includes six small commits, but the project requires the target branch to contain a single commit representing the completed change. Which merge method should you select?

A. Create a merge commit

B. Rebase and merge

C. Squash and merge

D. Close the pull request

ANSWER: C

Explanation:

Squash and merge is the appropriate method because it consolidates all commits included in the pull request into one new commit when the pull request is merged into its base branch. In this scenario, the contributor's six small commits may represent intermediate work—such as incremental implementation, corrections, or review updates—while the project's desired history is one clean, complete change on the target branch. The resulting commit contains the combined file changes from the pull request, providing a concise and readable history without losing the final implementation.

GitHub lets repository administrators enable squash merging and configure the default commit-message format used for the resulting commit. This makes the approach especially useful for projects that prefer each pull request to correspond to one logical commit in the target branch. The pull request itself remains available as a record of the discussion, review activity, and original commit sequence, while the base branch receives the single consolidated commit. For details, see About pull request merges and Merging a pull request.

QUESTION NO: 5

A developer commits `config.local.json` to a repository. The team later adds `config.local.json` to the repository's `.gitignore` file. What happens to the file?

A. Git removes the file from the repository at the next commit.

B. Git continues to track the file because it was already committed.

C. Git encrypts the file before the next push to GitHub.

D. Git ignores the file only when it is changed on the default branch.

ANSWER: B

Explanation:

`Git continues to track the file because it was already committed.` Git records files in the repository's index when they are added and committed. Because `config.local.json` was already committed before its ignore pattern was introduced, it is an existing tracked file. Adding an entry to `.gitignore` does not alter that tracked state, remove the file from prior or future commits, or suppress subsequent modifications to it from Git status.

Ignore rules are intended to prevent matching *untracked* files from being proposed for addition to a repository. This commonly applies to machine-specific configuration, generated output, local secrets files, and other artifacts that should remain outside version control from the outset. To retain the file on each contributor's machine while stop tracking it in the repository, a contributor must explicitly remove it from Git's index, such as by running `git rm --cached config.local.json`, then commit that removal together with the `.gitignore` change. GitHub's documentation explicitly notes that a file already tracked by Git is not affected merely by adding it to `.gitignore`. See Ignoring files and Removing sensitive data from a repository.

QUESTION NO: 6

Which attributes are displayed on a GitHub milestone page? Select two answers.

A. each milestone ' s completion percentage

B. a list of the open and closed issues associated with the milestone

C. a summary list of GitHub Projects tagged to the milestone

D. GitHub Action workflows that need to be executed to complete the milestone

ANSWER: A B

Explanation:

Each milestone ' s completion percentage is displayed because GitHub milestones provide an at-a-glance progress indicator for the work assigned to the milestone. GitHub calculates this progress from the proportion of associated issues and pull requests that have been closed, allowing maintainers and contributors to monitor how close the milestone is to completion without manually tallying completed work. This is particularly useful for release targets, iterations, and other deadline-driven goals.

A list of the open and closed issues associated with the milestone is also available from the milestone page. Milestones group related work items in one place, enabling a team to review what remains open and what has been completed. GitHub's milestone views support filtering the associated work by status, which helps users focus on outstanding items or review completed items while tracking progress toward the milestone's due date.

For more information, see GitHub Docs: About milestones and Creating and editing milestones.

QUESTION NO: 7

Which three aspects of a GitHub Codespaces development environment can be configured? Each answer represents a complete solution.

A. The default region for Codespace data storage

B. Shell preferences and scripts using dotfiles

C. The inactivity timeout period for the Codespace

D. Extensions and plugins for the development environment

E. Programming languages and tools required for the project

ANSWER: B D E

Explanation:

Shell preferences and scripts using dotfiles, **Extensions and plugins for the development environment**, and **Programming languages and tools required for the project** can be configured to create a productive and consistent GitHub Codespaces experience. A user can specify a dotfiles repository in their Codespaces settings. During Codespace creation, GitHub clones that repository and runs its installation script, enabling personal shell configuration, command aliases, Git configuration, and other initialization tasks.

For repository-level setup, a dev container configuration defines the containerized development environment used by Codespaces. The configuration can declare Visual Studio Code extensions through its customization settings, so contributors receive required language support, debuggers, formatters, linters, and other editor tooling when opening the project. It can also install the languages, SDKs, utilities, and project tools needed to build, test, and run the application. These components may be supplied through a base image, Dockerfile, Dev Container Features, or lifecycle commands. This combination allows a repository to provide repeatable project tooling while users retain personal environment preferences through dotfiles. See GitHub Docs: Introduction to dev containers and GitHub Docs: Personalizing GitHub Codespaces.

QUESTION NO: 8

Which two capabilities are advantages of the current GitHub Projects experience compared with GitHub Projects Classic?

Each answer presents a complete solution. Choose two.

A. GitHub Projects has multiple layout views.

B. GitHub Projects has Insights.

C. GitHub Projects are Copilot enabled.

D. GitHub Projects can be connected to third-party tools.

ANSWER: A B

Explanation:

GitHub Projects has multiple layout views. A single project can present the same project items and custom-field data in table, board, and roadmap layouts. The table layout supports structured planning and sorting across fields, the board layout supports workflow visualization through configurable columns, and the roadmap layout helps teams plan work over dates and iterations. Because these are views of one shared project rather than separate boards, a team can adapt the presentation to planning, triage, and reporting needs without duplicating project data. GitHub documents these supported layouts in Customizing the layout of a view.

GitHub Projects has Insights. Insights lets project members create configurable charts from project items and fields, enabling teams to visualize information such as item counts by status, priority, assignee, repository, or iteration. These charts can be saved as views and shared with collaborators, so stakeholders can monitor progress and trends from within the project. This built-in reporting capability supports clearer status tracking and project decision-making. GitHub explains the available chart-based reporting functionality in About insights for projects.

QUESTION NO: 9

Which statements describe characteristics of GitHub Teams? Select two answers.

A. Team visibility can be visible or secret.

B. Teams can be nested.

C. Teams has a maximum size of 100 users.

D. Teams requires access to at least one repository.

ANSWER: A B

Explanation:

Team visibility can be visible or secret. GitHub organizations can set a team's visibility to visible or secret. A visible team can be viewed and mentioned by all organization members, whereas a secret team is visible only to its members and organization owners. This setting lets organizations use teams both for broadly discoverable groups and for groups whose membership or purpose should not be exposed to everyone in the organization. Team visibility is separate from repository permissions, so it supports both collaboration and access-management scenarios.

Teams can be nested. GitHub supports parent and child teams within an organization. A child team can be created beneath a parent team to represent organizational structures such as a department, product group, and specialized subteams. Child teams inherit the members of their parent team, allowing organizations to manage membership and repository access more consistently while reducing repeated administrative work. Nested teams are especially useful when broad access needs to apply across a larger group while more specific permissions are managed for individual subgroups. For details, see GitHub's About teams documentation and its guidance on creating child teams.

QUESTION NO: 10

Which activities are counted on a user's GitHub profile contribution graph? Choose two.

A. commits made

B. public repositories

C. followers

D. issues closed

E. pull requests created

ANSWER: A E

Explanation:

commits made are counted on a GitHub profile's contribution graph when they meet GitHub's eligibility requirements. In particular, the commit must be associated with an email address connected to the contributor's account, and it generally must be made on the repository's default branch or `gh-pages` branch. Contributions from forks have additional requirements. Eligible commits provide a record of code changes made over time and can be displayed for public repositories; private-repository activity can be included as anonymized private contributions if the user enables that profile setting.

pull requests created are also a supported contribution event. Opening a pull request reflects work proposed for review and collaboration, so GitHub records it as contribution activity. Pull request contributions are attributed to the account that opened the pull request and appear according to the relevant repository visibility and profile contribution settings. Together, commits and pull requests represent two core forms of collaborative development activity that GitHub uses to populate the profile graph. GitHub lists qualifying contribution events and their conditions in its Profile contributions reference. For details on how public and private activity is presented on the profile, see Viewing contributions on your profile.

QUESTION NO: 11

What is the main purpose of creating a security policy for a GitHub repository?

A. To ensure that peer code review occurs before new changes are merged

B. To define which types of secrets are blocked with push protection

C. To describe how security vulnerabilities should be responsibly disclosed

D. To customize the repository's Dependabot configuration

ANSWER: C

Explanation:

The primary purpose is **to describe how security vulnerabilities should be responsibly disclosed**. A repository security policy gives security researchers, users, and contributors clear instructions for reporting a potential vulnerability privately to the project maintainers. On GitHub, this policy is typically provided in a `SECURITY.md` file. It can specify supported versions of the project, a preferred reporting channel such as a private security advisory or email address, expected response times, and guidance on coordinated disclosure. Providing these details helps reporters avoid disclosing a vulnerability publicly before maintainers have had an opportunity to investigate and develop a fix. It also establishes an agreed process that improves communication and helps protect people who use the repository. When GitHub detects a security policy in a supported location, it makes the policy discoverable through the repository's Security area, so potential reporters can readily find the project's reporting instructions. GitHub's documentation explains how to add and structure a repository security policy, including use of the `SECURITY.md` file: Adding a security policy to your repository. For broader context on GitHub's security capabilities and vulnerability-management workflows, see GitHub security features.

QUESTION NO: 12

Which keywords can you use with an issue reference in a pull request description to link the pull request to the issue and automatically close that issue when the pull request is merged? Choose three.

A. fix

B. closed

C. merge

D. connects

E. resolves

F. join

ANSWER: A B E

Explanation:

`fix`, `closed`, and `resolves` are valid GitHub closing keywords. Use one of these terms together with an issue reference in the pull request description, such as `Fix #123`, `Closed #123`, or `Resolves #123`. GitHub then displays the relationship between the pull request and the issue, providing clear traceability from the tracked work item to the proposed code change. When the pull request is merged into the repository's default branch, GitHub automatically closes the referenced issue. This workflow helps teams keep issue status aligned with completed work and avoids a separate manual step to close the issue. GitHub supports keyword variants in the close, fix, and resolve families; the selected terms are recognized variants from those supported families. The issue reference can use the issue number, for example `#123`, or a cross-repository reference when appropriate. For the automatic-closing behavior to apply, the pull request must target the repository's default branch. See GitHub's documentation for linking pull requests to issues and its complete list of supported closing keywords.

QUESTION NO: 13

When multiple collaborators are assigned to the same issue or pull request, what is the result?

A. The issue or pull request is automatically marked as “In Progress.”

B. The issue or pull request is locked for editing by others.

C. Only the first collaborator is notified.

D. All assigned collaborators are notified and share responsibility.

ANSWER: D

Explanation:

All assigned collaborators are notified and share responsibility. In GitHub, assignees identify the people expected to take ownership of, contribute to, investigate, review, or otherwise follow up on an issue or pull request. Adding more than one assignee makes that shared ownership visible to everyone with access to the work item, which is useful when completing the work requires coordination among several people. Each assigned user is associated with the issue or pull request and receives assignment-related notifications, subject to that user’s notification settings. Assignment therefore serves as a clear collaboration and accountability signal rather than simply naming a single primary owner. GitHub supports assigning multiple people to an individual issue or pull request—up to ten assignees—provided the users have the appropriate repository access. This helps teams communicate responsibility directly in the repository and gives each participant a clear connection to the work that needs attention. For details, see GitHub Docs: Assigning issues and pull requests to other GitHub users and GitHub Docs: About issues.

QUESTION NO: 14

Which term best describes Markdown?

A. Lightweight markup language

B. Programming language

C. Scripting language

D. Version control system

E. Containerization solution

ANSWER: A

Explanation:

Lightweight markup language is correct because Markdown is a plain-text syntax used to apply structural formatting to written content while keeping the source easy for people to read and edit. Its concise conventions let an author represent headings, lists, emphasis, links, images, quotations, and code without using a visual editor or writing verbose HTML. For example, hash symbols create headings, asterisks can add emphasis, and bracket-and-parenthesis syntax creates links.

GitHub renders Markdown throughout its collaboration experience, including README files, issues, pull requests, discussions, wikis, and comments. This makes it particularly useful for repository documentation because the formatted content can be stored as text alongside code, reviewed in commits, and rendered clearly on GitHub. GitHub Flavored Markdown also provides features that support collaborative technical writing, such as tables, task lists, and code blocks. GitHub’s documentation describes Markdown as a way to add formatting to plain-text documents and identifies the locations where it is supported. See Writing on GitHub and Basic writing and formatting syntax.

QUESTION NO: 15

Which GitHub destination can you use to discover, browse, and install tools that extend your development workflow?

A. GitHub Marketplace

B. GitHub Apps

C. Organization settings

D. Explore

ANSWER: A

Explanation:

GitHub Marketplace is the centralized destination for finding and installing tools that extend GitHub and support software-development workflows. It provides searchable, categorized listings for GitHub Apps and GitHub Actions, allowing users and organizations to locate integrations for common needs such as continuous integration, project management, code quality, security scanning, monitoring, and deployment. A Marketplace listing presents information needed to evaluate a tool, including its purpose, publisher, permissions, pricing where applicable, and installation or setup steps. Users can then install an app for a personal account or organization, subject to the relevant authorization requirements, or use an action in a workflow. This discovery-to-installation experience is specifically what GitHub Marketplace is designed to provide. GitHub's documentation describes Marketplace as a place to discover apps and actions that can improve development workflows, and it documents how GitHub Apps can be installed directly from their Marketplace listings. For more information, see About GitHub Marketplace and Installing a GitHub App from GitHub Marketplace.

QUESTION NO: 16

Which action highlights a repository discussion by keeping it at the top of the Discussions page?

A. Save the discussion.

B. Pin the discussion.

C. Create an issue from the discussion.

D. Star the discussion.

ANSWER: B

Explanation:

Pin the discussion. is the correct action because GitHub uses pinned discussions to keep important conversations prominently visible at the top of a repository's Discussions page. This is useful when maintainers need community members to readily find information that remains relevant over time, such as an announcement, a welcome post, contribution guidance, a feedback thread, or frequently referenced resources. Rather than being positioned according to normal discussion activity, a pinned discussion stays in the dedicated pinned area, making it easier for visitors to locate even as new discussions are created.

People with the required repository permissions can pin a discussion from its menu and can later unpin it when it no longer needs special visibility. GitHub also permits multiple discussions to be pinned, enabling maintainers to organize and emphasize several key community topics simultaneously. Pinning is specifically designed for promoting an existing discussion within the repository's Discussions experience, so it directly meets the requirement to place the post at the top of that page. For the documented procedure and permission details, see GitHub Docs: Managing discussions. For an overview of repository discussions, see GitHub Docs: About discussions.

QUESTION NO: 17

An organization within a GitHub enterprise has its base repository permission set to **No permissions**. Which users can view every internal repository owned by that organization?

(Each answer is a complete solution. Choose two.)

A. outside collaborators in the same organization

B. outside collaborators with repo Admin access

C. members in the same organization

D. members in another organization in that enterprise

ANSWER: C D

Explanation:

members in the same organization and **members in another organization in that enterprise** can view internal repositories because internal visibility is scoped to the enterprise, not solely to the organization that owns the repository. GitHub describes an internal repository as one that is accessible to all members of the enterprise. This model supports sharing source code, documentation, and common tooling between organizations that belong to the same enterprise without needing to grant access to each repository individually.

The organization's base permission of **No permissions** does not change this enterprise-wide visibility characteristic for internal repositories. Base permissions determine the default access granted to organization members for repositories in that organization, while internal repository visibility gives authenticated enterprise members read access by default. Therefore, both a member of the repository-owning organization and a member of another organization under the same enterprise are included in the audience that can see all internal repositories in that organization.

GitHub documents the enterprise access model for internal repositories in About repository visibility and provides related visibility guidance in Setting repository visibility.