Accelerate Your Certification with Microsoft GH-600 Practice Questions

Last chance to prepare smart! Get your hands on free Microsoft Developing in Agentic AI Systems exam PDF questions. Study real GH-600 dumps with verified answers and fast-track your certification success with PrepBolt GH-600 exam pdf questions and answers.

Thank you for Downloading GH-600 exam PDF Demo

https://prepbolt.com/GH-600.html

QUESTIONS & ANSWERS

DEMO VERSION

(LIMITED CONTENT)

Question 1

Question Type: MultipleChoice

Case Study: Mix Questions

Mix Questions

GH-600 Mix Questions IN THIS CASE STUDY

You have a GitHub Enterprise organization that has Copilot memory enabled.

You create a new repository.

What are two ways that memories will be deleted from the repository? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

Options:

A- manually by repository members
B- when the repository is archived
C- when the code that created the memory is deleted
D- automatically after 28 days
E- when pull requests are merged

Answer:

A, D

Explanation:

GitHub Copilot repository memories can be removed manually and through automatic retention cleanup. Repository owners or administrators can review repository-level facts under the repository's Copilot Memory settings and delete stored facts that are obsolete, misleading, or no longer appropriate. This makes A a valid deletion mechanism.

D is also correct. GitHub applies an inactivity-based retention policy: a stored repository fact or user preference that goes unused for 28 days is automatically deleted. The 28-day period can restart when Copilot successfully validates and uses that memory, so this should be understood as an unused-memory expiration period rather than an unconditional deletion exactly 28 days after creation.

Deleting the source code referenced by a memory does not directly delete that memory. Repository-level facts retain citations and are validated against the current branch before use; if the supporting information is no longer valid, Copilot does not use the fact. Likewise, archiving a repository or merging a pull request is not documented as an automatic memory-deletion event.

Study Guide Reference Topics: Manage Memory, State, and Execution; repository memory lifecycle; memory validation; memory retention and deletion; persistent agent knowledge.

Question 2

Question Type: MultipleChoice

Case Study: Mix Questions

Mix Questions

GH-600 Mix Questions IN THIS CASE STUDY

You have a GitHub Enterprise repository.

An agent opens pull requests to the main branch.

You need to ensure that changes to .github/workflows/* and /infra/* require approval from designated reviewers before merge.

What should you configure?

Options:

A- a ruleset and an agents.md file
B- a ruleset and a .copilotignore file
C- a branch protection rule and a CODEOWNERS file
D- a branch protection rule and copilot-instructions.md

Answer:

C

Explanation:

The correct solution combines a branch protection rule with a CODEOWNERS file. CODEOWNERS allows the repository to associate specific paths with designated users or teams. For example, entries can assign security or platform reviewers to .github/workflows/* and /infra/*. When a pull request modifies those paths, GitHub automatically identifies the corresponding code owners.

The enforcement mechanism comes from branch protection on main. Configure the protection rule to Require a pull request before merging and enable Require review from Code Owners. GitHub then blocks the merge until an applicable code owner approves the affected files. This converts CODEOWNERS from simple reviewer routing into an enforceable merge-control boundary.

agents.md and copilot-instructions.md provide behavioral guidance to AI agents; they do not enforce reviewer authorization. .copilotignore likewise does not establish mandatory merge approval. Although GitHub rulesets can also implement review controls, none of the ruleset choices provides the required CODEOWNERS pairing.

Study Guide Reference Topics: Implement Guardrails and Accountability; protected branches; required human review; CODEOWNERS; repository governance; approval gates.

Question 3

Question Type: MultipleChoice

Case Study: Mix Questions

Mix Questions

GH-600 Mix Questions IN THIS CASE STUDY

An agentic coding session in the terminal has grown too long, and you want to discard all prior conversation history and start clean, without summarization. Which slash command should you run?

Options:

A- /compact
B- /clear
C- /reset-context
D- /usage

Answer:

B

Explanation:

The correct command is /clear. In GitHub Copilot CLI, /clear starts a fresh conversational state by removing the current chat history from the active session. This is the appropriate action when the requirement is to discard prior conversation context completely rather than compress or summarize it.

/compact is specifically designed for context management during long-running sessions. It summarizes older conversation history so that useful information can be retained while freeing context-window capacity. Because the question explicitly says ''without summarization,'' /compact does not satisfy the requirement.

/usage displays session consumption and usage information rather than modifying conversation state.

/reset-context is not the standard slash command used for clearing Copilot CLI conversation history in this scenario.

The operational distinction is important in agentic workflows. Compaction preserves semantic continuity, while clearing abandons the prior conversational state and gives the agent a clean starting point. Clearing is therefore appropriate when accumulated context has become irrelevant, misleading, or overly large and you deliberately do not want previous reasoning, instructions, or task history carried forward.

Study Guide Reference Topics: Manage Memory, State, and Execution; conversation-state management; context-window control; session reset versus context compaction.

Question 4

Question Type: MultipleChoice

Case Study: Mix Questions

Mix Questions

GH-600 Mix Questions IN THIS CASE STUDY

You want to prevent GitHub Copilot from ever suggesting completions or making edits inside a directory containing sensitive credentials templates. What should you configure?

Options:

A- A .copilotignore file
B- A CODEOWNERS file
C- Branch protection rules
D- A repository ruleset

Answer:

A

Explanation:

A is the intended answer among the supplied choices, because the required control is a Copilot-specific content exclusion mechanism rather than Git authorization or branch governance. The important technical concept is excluding the sensitive directory from Copilot's usable context and suggestions.

Current GitHub documentation implements this through Copilot Content exclusion settings at

repository, organization, or enterprise scope. Administrators can specify exact files, directory paths, and patterns to exclude. For excluded content, GitHub states that inline suggestions are unavailable in the affected files, excluded file content does not inform suggestions elsewhere, and the files are excluded from supported Copilot responses and code review.

Neither CODEOWNERS nor branch protection prevents Copilot from reading or generating suggestions for a particular directory. CODEOWNERS establishes reviewers/ownership, while branch protections and rulesets govern repository operations such as pushes, merges, and review requirements. Those mechanisms operate too late in the lifecycle to satisfy a requirement concerning Copilot's access to sensitive file content.

For exam purposes, therefore, A maps to the content-exclusion concept. In current GitHub administration, implement the control through Settings Copilot Content exclusion and specify the sensitive directory path.

Study Guide Reference Topics: Implement Guardrails and Accountability; sensitive-context exclusion; least exposure; repository security boundaries.

Question 5

Question Type: MultipleChoice

Case Study: Mix Questions

Mix Questions

GH-600 Mix Questions IN THIS CASE STUDY

Your team wants Copilot's suggestions to reflect knowledge of internal library APIs that are not publicly documented and not present in the codebase being edited. What is the most appropriate solution?

Options:

A- Add the internal docs as copilot-instructions.md
B- Connect an MCP server that exposes the internal documentation
C- Use .copilotignore to hide public APIs
D- Enable plan mode

Answer:

B

Explanation:

An MCP server exposing the internal documentation is the appropriate solution because Model Context Protocol extends Copilot with information and capabilities located outside the repository's native context. GitHub describes MCP as a mechanism for integrating Copilot with external systems, enabling it to obtain context or invoke functionality that would otherwise be unavailable from the codebase alone.

This architecture is appropriate for proprietary library documentation because the internal API knowledge can remain centrally maintained rather than being duplicated into every repository. The MCP integration can expose a controlled documentation lookup/search capability, allowing Copilot to retrieve relevant definitions, usage patterns, or internal API information when required.

Option A is inferior because copilot-instructions.md is designed for concise repository-specific instructions and conventions, not as a substitute for a potentially large external documentation corpus. GitHub recommends it for persistent guidance such as building, testing, and repository conventions. Option C would remove context rather than add proprietary knowledge. Plan mode affects workflow sequencing and does not provide new external information.

Study Guide Reference Topics: Implement Tool Use and Environment Interaction; MCP-based context augmentation; external knowledge integration; tool-mediated retrieval.

==========

Question 6

Question Type: MultipleChoice

Case Study: Mix Questions

Mix Questions

GH-600 Mix Questions IN THIS CASE STUDY

You are architecting an agentic AI system and need the agent's tool-calling behavior to be constrained so it can only call a specific allow-listed set of MCP tools, never arbitrary ones. What should you configure?

Options:

A- Tool/server allow-list in the MCP client configuration
B- .copilotignore
C- Repository ruleset
D- /usage

Answer:

A

Explanation:

The correct answer is A because tool availability must be constrained at the agent/MCP integration boundary. GitHub custom-agent configuration provides a tools property that explicitly determines which tools the agent is permitted to use, including tools supplied by configured MCP servers. Rather than enabling all tools with ['*'] or omitting the restriction, administrators can specify individual tool names or aliases and thereby implement an explicit allow-list.

This is a fundamental least-privilege control for agentic systems. An MCP server can expose numerous capabilities, but exposure by the server does not mean every capability should automatically be available to every agent. Restricting the agent to the tools required for its assigned role reduces unintended actions and limits the blast radius of erroneous reasoning, prompt injection, or excessive autonomy. GitHub also supports repository MCP configuration through which administrators determine which external MCP integrations are available.

.copilotignore does not implement MCP authorization. Repository rulesets govern GitHub repository operations such as branches and merges, while /usage reports runtime/session information.

Study Guide Reference Topics: Implement Tool Use and Environment Interaction; MCP integration; tool allow-listing; least privilege; execution boundaries.

==========

Thank You for trying GH-600 PDF Demo

To try our GH-600 practice exam software visit link below

https://prepbolt.com/GH-600.html

Start Your GH-600 Preparation

Use Coupon “SAVE50” for extra 50% discount on the purchase of Practice Test Software. Test your GH-600 preparation with actual exam questions.