CERTLABZ CERTIFICATION BOARD

Certified Git and Version Control Specialist (CGVS)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working Git and version control specialist would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1

Domain 1: Version Control Foundations

DATA

Emma Wilson compares two version control models, with one repository trait missing.

Model	Repository copy
Centralized	Single server copy
Distributed	(to choose)
Local only	No shared history
File share	Manual folder copies

Table 1.1 Version control models.

What does the distributed model give each developer?

A. Server only copy
B. Full local copy
C. One shared copy
D. No local history

Correct answer: B
A distributed model gives every developer a full local copy of the repository, including its complete history, so most work happens offline. A centralized model keeps one copy on the server, and the other options describe having no usable local history.

Question 2

Domain 2: Git Fundamentals

CONFIG

James Carter runs a short sequence and then checks the status of the file.

```
$ git init
$ git add app.js
$ git status
  Changes to be committed:
    new file: app.js
```

Listing 2.1 A new file after add.

What state is app.js in?

A. Committed to history
B. Untracked entirely
C. Staged for commit
D. Ignored by Git

Correct answer: C
Running git add moves the file into the staging area, so app.js is staged for commit and waiting for the next git commit. It is not yet committed to history, and it is neither untracked nor ignored once it has been added.

Question 3

Domain 3: Branching and Merging

CONFIG

Olivia Davis merges a feature branch back into main and sees the merge type.

```
$ git checkout -b feature
$ git commit -m work
$ git checkout main
$ git merge feature
  Fast-forward
```

Listing 3.1 A fast-forward merge.

Why did the merge fast-forward?

A. Main had no commits
B. History was rewritten
C. A conflict occurred
D. Feature was reverted

Correct answer: A
A fast-forward happens because main gained no new commits after the branch was created, so Git simply moves the main pointer forward to the feature tip. When the two branches have diverged instead, Git records a separate merge commit.

Question 4 Domain 4: Collaboration and Remotes DATA

Liam Anderson maps each remote command to its effect, with one effect missing.

Command	Effect
git fetch	Download, do not merge
git pull	(to choose)
git push	Upload local commits
git remote -v	List remote URLs

Table 4.1 Remote commands and effects.

What does git pull do?

A. Delete remote branch
B. Only list remotes
C. Rewrite all history
D. Fetch then merge

Correct answer: D
A git pull downloads new commits from the remote and then merges them into the current branch, combining a fetch and a merge in one step. It does not delete branches, only list remotes, or rewrite history, which are the effects of other commands.

Question 5 Domain 5: History, Recovery and Advanced Git CONFIG

Ava Thompson runs a hard reset by mistake and then inspects the reference log.

```
$ git reset --hard HEAD~1
$ git reflog
  a1b2c3d HEAD@{1}: commit: fix
  e4f5g6h HEAD@{0}: reset: moving
```

Listing 5.1 Reflog after a reset.

How can the lost commit be recovered?

A. Delete the reflog
B. Clone again fresh
C. Use HEAD@{1}
D. Run git gc now

Correct answer: C
The reflog records where HEAD pointed before the reset, so resetting or checking out to HEAD@{1} restores the commit that seemed lost. Cloning again or running garbage collection does not bring back local work that only the reflog still references.

Question 6 — Domain 6: Workflows, CI and Governance — DATA

David Brown maps each workflow to its key trait, with one trait missing.

Workflow	Key trait
Git flow	Release and hotfix branches
Trunk based	(to choose)
Feature branch	One branch per feature
Forking	Contributor owns a fork

Table 6.1 Workflows and traits.

What is the key trait of trunk based development?

A. One long release branch
B. Frequent small merges
C. No shared main branch
D. Branch per big release

Correct answer: B
Trunk based development has everyone integrate frequent small merges into a single shared trunk, keeping branches short lived to reduce painful merges. The other options describe long lived branches or the absence of a shared main, which run counter to the approach.

Question 7 — Domain 2: Git Fundamentals

A build keeps producing a large log file that a developer does not want Git to track, yet it shows up as untracked every time. What best keeps it out of commits?

A. Commit it again
B. Use a .gitignore
C. Push more often
D. Run git status often

Correct answer: B
Listing the log file pattern in a .gitignore tells Git to leave the untracked artifact alone, so it stops appearing as a candidate for commits. Committing it again would add the noise to history, and pushing or checking status more often does nothing to exclude it.

Question 8 Domain 3: Branching and Merging

Two developers edited the same lines, and a merge stops with conflict markers written into the file. What must happen before the merge can complete?

A. Resolve then commit
B. Force push to main
C. Delete both branches
D. Run git fetch again

Correct answer: A
A conflict means Git could not choose between the two edits, so the developer must resolve the marked sections by hand and then commit to finish the merge. Force pushing, deleting branches, or fetching again does not settle the conflicting lines.

Question 9 Domain 4: Collaboration and Remotes

A contributor cannot push to an upstream project because they lack write access, but they still want their change reviewed and merged. What is the standard open source path?

A. Push straight to main
B. Email a zip file
C. Delete upstream repo
D. Fork and open a PR

Correct answer: D
Without write access, the contributor forks the repository, pushes to their own fork, and opens a pull request for the maintainers to review and merge. Pushing straight to main is blocked, and emailing files or touching the upstream repo is not how hosted collaboration works.

Question 10 Domain 6: Workflows, CI and Governance

A team wants every pull request tested automatically before it can merge, so that broken code never reaches the main branch. What best enforces this?

A. Longer review meetings
B. A required CI check
C. More manual testing
D. Larger merge commits

Correct answer: B
A required continuous integration check runs the test suite on each pull request and blocks the merge until it passes, keeping broken code off the main branch. Longer meetings, extra manual testing, or larger commits do not provide the same automatic gate.

CGVS-001. https://certlabz.com/certifications/certified-git-and-version-control-specialist.html