GIT

---

1. Which command initializes a new Git repository?

A. git start
B. git init
C. git create
D. git new
Answer: B
Explanation: ‘git init‘ creates a new ‘.git‘ directory in the current folder, preparing it to track version control history.

2. What does ‘git clone‘ do?

A. Create a duplicate branch
B. Copy remote repository to local machine
C. Duplicate commit
D. Clone stash
Answer: B
Explanation: ‘git clone‘ downloads an entire repository (history and working tree) from a remote source to your local environment.

3. Which command stages all modified files for commit?

A. git add .
B. git commit -a
C. git stage --all
D. git push
Answer: A
Explanation: ‘git add .‘ stages tracked and new files in the current directory and below, preparing them for the next commit.

4. How do you view commit history?

A. git history
B. git log
C. git commits
D. git reflog
Answer: B
Explanation: ‘git log‘ lists commits in reverse chronological order, showing hashes, authors, dates, and messages.

5. What does ‘git status‘ display?

A. Current branch, staged and unstaged changes

B. Remote repository status
C. Commit history
D. Branch history
Answer: A
Explanation: ‘git status‘ summarizes the working tree state, showing staged files, unstaged changes, and untracked files.

6. Which command creates a new branch called feature?

A. git branch feature
B. git create branch feature
C. git checkout feature
D. git branch -n feature
Answer: A
Explanation: ‘git branch feature‘ creates the branch without switching to it, allowing you to begin development on a new line.

7. How do you switch to an existing branch named dev?

A. git switch dev
B. git checkout dev
C. Both A and B (Git 2.23+ supports switch)
D. git change dev
Answer: C
Explanation: Older versions use ‘git checkout‘ while newer Git versions introduced ‘git switch‘ as a clearer alternative for branch changes.

8. What does ‘git merge feature‘ do?

A. Merges feature branch into current branch
B. Deletes feature branch
C. Rebase branch
D. Clone feature
Answer: A
Explanation: ‘git merge feature‘ integrates changes from the ‘feature‘ branch into the branch you currently have checked out.

9. Which command discards local changes permanently (unstaged)?

A. git checkout -- file
B. git restore file
C. git reset --hard
D. Options A and B (new command) revert modifications in working tree
Answer: D
Explanation: ‘git checkout -- file‘ (legacy) and ‘git restore file‘ (new) both revert file changes back to the last committed state without touching staging.

10. What does ‘git commit -m "message"‘ do?

A. Commits staged changes with message
B. Commits all changes automatically
C. Commits only tracked files
D. Pushes commits

Answer: A

Explanation: Commits capture staged changes and attach a message describing the change, forming a new snapshot in history.

11. How do you add a remote repository named origin?

A. git remote add origin <url>
B. git add remote origin <url>
C. git remote origin <url>
D. git clone origin <url>

Answer: A

Explanation: ‘git remote add origin URL‘ defines a remote repository named origin so you can fetch and push to it.

12. Which command uploads local commits to remote?

A. git push
B. git upload
C. git publish
D. git sync

Answer: A

Explanation: ‘git push‘ sends local commits from your branch to its remote-tracking branch, updating the remote repository.

13. What does ‘git pull‘ perform by default?

A. Fetch then merge from remote tracking branch
B. Fetch only
C. Merge only
D. Rebase only

Answer: A

Explanation: ‘git pull‘ equals ‘git fetch‘ followed by ‘git merge‘ of the remote branch into your current branch unless configured otherwise.

14. How do you fetch remote updates without merging?

A. git fetch
B. git pull --no-merge
C. git update
D. git sync remote

Answer: A

Explanation: ‘git fetch‘ downloads new commits and references from remotes without integrating them into local branches.

15. Which command shows differences between working tree and staging area?

A. git diff
B. git diff --staged
C. git status
D. git compare

Answer: A

Explanation: By default ‘git diff‘ compares the working directory against the index (staging area), highlighting unstaged changes.

16. How do you view differences between staged changes and last commit?

A. git diff --cached
B. git diff --staged
C. Both A and B (aliases)
D. git diff HEAD

Answer: C

Explanation: Both flags show the diff between the index and the last commit (HEAD), previewing what will be committed.

17. What does ‘git reset HEAD file‘ do?

A. Unstage file while keeping working tree changes
B. Remove file
C. Delete commit
D. Reset branch

Answer: A

Explanation: ‘git reset HEAD file‘ moves the file out of the index, so changes remain in the working directory but are no longer staged.

18. Which command amends the last commit message?

A. git commit --amend
B. git amend
C. git fix commit
D. git change commit

Answer: A

Explanation: ‘git commit --amend‘ opens the last commit for editing, allowing you to alter the message and staged content.

19. What is the effect of ‘git stash‘?

A. Save working directory and index state for later use
B. Delete branch
C. Merge changes

D. Create tag

Answer: A

Explanation: Stashing stores dirty working state on a stack so you can revert to a clean state and reapply changes later.

20. How do you apply the most recent stash and keep it?

A. git stash apply

B. git stash pop

C. git stash drop

D. git stash show

Answer: A

Explanation: ‘git stash apply‘ applies stash contents but retains the stash entry; ‘pop‘ applies and removes it.

21. Which command lists available stashes?

A. git stash list

B. git stash show

C. git stash log

D. git list stash

Answer: A

Explanation: ‘git stash list‘ shows stash entries, each with an index and description for reference.

22. What does ‘git tag -a v1.0 -m "release"‘ create?

A. Annotated tag named v1.0

B. Lightweight tag

C. Branch

D. Release branch

Answer: A

Explanation: ‘-a‘ creates an annotated tag with metadata and message, stored as a full object in Git history.

23. How do you delete a local branch named feature after merge?

A. git branch -d feature

B. git branch --delete feature

C. Both A and B

D. git delete branch feature

Answer: C

Explanation: ‘git branch -d‘ (or ‘--delete‘) removes a local branch, warning if unmerged commits remain.

24. Which command removes a remote branch?

A. git push origin --delete feature

B. git delete origin feature
C. git branch -dr origin/feature
D. Both A and C (C removes tracking reference)
Answer: D
Explanation: ‘git push origin --delete‘ deletes the remote branch; ‘git branch -dr‘ removes the local remote-tracking reference.

25. What is the default name for initial branch in newer Git versions?

A. master
B. main
C. trunk
D. default
Answer: B
Explanation: Many setups now default to ‘main‘, though configuration or environment may still use ‘master‘ unless changed.

26. Which command sets email globally?

A. git config --global user.email "you@example.com"
B. git set email
C. git username email
D. git email config
Answer: A
Explanation: Setting ‘user.email‘ globally applies to all repositories the user commits to, embedding the address in commit metadata.

27. How do you configure Git to use rebase on pull?

A. git config --global pull.rebase true
B. git pull --rebase-only
C. git rebase pull true
D. git config pull rebase yes
Answer: A
Explanation: Configuring ‘pull.rebase‘ ensures ‘git pull‘ uses rebase instead of merge by default, keeping history linear.

28. What does ‘git cherry-pick <commit>‘ do?

A. Apply changes introduced by specific commit onto current branch
B. Delete commit
C. Rebase entire branch
D. Merge branch
Answer: A
Explanation: Cherry-picking duplicates the changes from a commit onto the current branch, useful for selective merging.

29. Which command displays all branches including remote?

A. git branch -a
B. git branch -r
C. Both (A includes local & remote)
D. git branches
Answer: C
Explanation: ‘git branch -a‘ lists both local and remote-tracking branches; ‘-r‘ lists remote only.

30. How do you rename current branch to dev?

A. git branch -m dev
B. git rename dev
C. git move dev
D. git checkout -b dev
Answer: A
Explanation: ‘git branch -m new-name‘ renames the current branch; ‘checkout -b‘ creates a new branch rather than renaming.

31. What does ‘git revert <commit>‘ achieve?

A. Create new commit that undoes specified commit
B. Delete commit history
C. Reset HEAD
D. Reapply commit
Answer: A
Explanation: Reverting records a new commit that reverses changes, preserving history without rewriting it.

32. Which command shows commits affecting a file?

A. git log -- file
B. git blame file
C. git history file
D. git show file
Answer: A
Explanation: ‘git log -- <file>‘ filters commits that touched a file, useful for tracking modifications.

33. How do you view who last modified each line?

A. git blame file
B. git annotate file
C. Both A and B (aliases)
D. git history file
Answer: C

Explanation: ‘git blame‘ (and alias ‘annotate‘) outputs line-by-line attribution, showing commit hashes and authors.

34. What does ‘git remote -v‘ list?

A. Remote names with fetch/push URLs
B. Remote branches
C. Tags
D. Submodules

Answer: A

Explanation: ‘git remote -v‘ prints configured remotes along with their fetch and push URLs, helping verify connections.

35. Which command updates submodules to latest commit on tracked branch?

A. git submodule update --remote
B. git submodule sync
C. git submodule init
D. git submodule pull

Answer: A

Explanation: ‘git submodule update --remote‘ fetches upstream changes for submodules and updates them to the referenced branch commits.

36. How do you stage part of a file?

A. git add -p file
B. git add --patch file
C. Both
D. git add --partial file

Answer: C

Explanation: Patch mode interactively selects hunks to stage, enabling fine-grained commits; ‘-p‘ is shorthand for ‘--patch‘.

37. What is the purpose of ‘.gitignore‘?

A. Specify files to be ignored by Git
B. Delete files
C. Track ignored files
D. Force add files

Answer: A

Explanation: ‘.gitignore‘ lists patterns for files Git should ignore, preventing them from being tracked or accidentally committed.

38. Which command shows remote-tracking branch references?

A. git branch -vv
B. git remote show origin
C. Both (A displays upstream branches)

D. git status -r

Answer: C

Explanation: ‘git branch -vv‘ shows local branches and their upstream; ‘git remote show origin‘ lists fetch/push branches and tracking status.

39. What does ‘git reflog‘ display?

A. History of HEAD moves and branch updates

B. Remote logs

C. Tag history

D. Merge logs

Answer: A

Explanation: The reflog records reference updates, enabling recovery of commits even after branch deletions or resets.

40. How do you create a signed commit?

A. git commit -S -m "message"

B. git commit --sign

C. git commit -sign

D. git commit -mS

Answer: A

Explanation: ‘-S‘ signs the commit with your configured GPG key, embedding a signature to verify authenticity.

41. Which command finds branches already merged into main?

A. git branch --merged main

B. git merge --list main

C. git branch -m main

D. git main --merged

Answer: A

Explanation: ‘git branch --merged main‘ lists branches whose tips are reachable from ‘main‘, indicating they can be safely deleted.

42. What does ‘git clean -fd‘ do?

A. Remove untracked files and directories

B. Remove tracked files

C. Reset repository

D. Clean commit history

Answer: A

Explanation: ‘git clean -fd‘ deletes untracked files (‘-f‘) and directories (‘-d‘), useful for restoring to a clean working tree before builds.

43. How do you configure alias for log?

A. git config --global alias.lg "log --oneline --graph"

B. git alias log
C. git log alias set
D. git config alias log
Answer: A
Explanation: Creating Git aliases via ‘git config‘ provides shortcuts; ‘alias.lg‘ sets ‘git lg‘ to run the specified log command.

44. Which command sets upstream for branch to origin/dev?

A. git push -u origin dev
B. git branch --set-upstream-to=origin/dev
C. git track origin/dev
D. Both A and B
Answer: D
Explanation: Pushing with ‘-u‘ sets upstream automatically; ‘git branch --set-upstream-to‘ manually associates a branch with its remote.

45. What is ‘.gitattributes‘ used for?

A. Define path-specific Git attributes like line endings
B. Ignore files
C. Manage hooks
D. Configure users
Answer: A
Explanation: ‘.gitattributes‘ controls behavior per path, such as enabling text normalization, diff drivers, or export rules.

46. How do you enable automatic CRLF conversion on checkout for Windows?

A. git config core.autocrlf true
B. git config core.autocrlf input
C. git config core.crlf true
D. git config core.windows true
Answer: A
Explanation: Setting ‘core.autocrlf true‘ converts LF to CRLF on checkout and back to LF on commit, preventing mixed line endings on Windows.

47. Which command lists tags?

A. git tag
B. git tag -l
C. Both
D. git show tags
Answer: C
Explanation: ‘git tag‘ without arguments lists available tags; ‘-l‘ is optional for listing with patterns.

48. What does ‘git describe‘ output?

A. Closest tag reachable from current commit
B. Repository description
C. Branch description
D. Commit message
Answer: A
Explanation: ‘git describe‘ uses tags and distance (number of commits) to produce human-readable version identifiers for commits.

49. How do you rebase current branch onto main?

A. git rebase main
B. git rebase origin/main
C. Both depending on tracking
D. git rebase --onto main
Answer: C
Explanation: Rebasing onto ‘main‘ uses your local copy; using ‘origin/main‘ rebases onto the remote-tracking branch, depending on workflow.

50. What is the effect of ‘git rebase -i HEAD~3‘?

A. Interactive rebase of last three commits
B. Rebase onto HEAD
C. Merge last three commits
D. Delete commits
Answer: A
Explanation: Interactive rebase lets you edit, reorder, squash, or drop the last three commits via an editor before rewriting history.

51. Which file contains per-repository configuration?

A. .git/config
B. ~/.gitconfig
C. /etc/gitconfig
D. .gitconfig.local
Answer: A
Explanation: Each repo stores settings in ‘.git/config‘, overriding user (‘~/.gitconfig‘) and system (‘/etc/gitconfig‘) configurations.

52. How do you check for merge conflicts remaining?

A. git status
B. git diff --name-only --diff-filter=U
C. Both
D. git conflict list
Answer: C

Explanation: ‘git status‘ highlights unmerged paths; the diff command filters for files with unresolved conflicts (‘U‘ status).

53. What does ‘git ls-tree HEAD‘ show?

A. Tree objects (files/directories) at commit HEAD
B. Branches
C. Logs
D. Tags

Answer: A

Explanation: ‘git ls-tree‘ lists the contents of a tree object (typically a commit), showing blobs and subtrees with their permissions and hashes.

54. How can you bisect to find a bad commit?

A. git bisect start; git bisect bad; git bisect good <commit>
B. git findcommit
C. git debug commits
D. git search bug

Answer: A

Explanation: ‘git bisect‘ performs binary search between known good and bad commits to identify the first offending commit.

55. What is the purpose of hooks in Git?

A. Custom scripts triggered by repository events
B. Manage branches
C. Configure remotes
D. Track files

Answer: A

Explanation: Hooks (e.g., pre-commit, post-receive) run scripts in ‘.git/hooks‘, automating checks, deployments, or notifications on Git events.

56. Which command clones repository with submodules initialized?

A. git clone --recurse-submodules <url>
B. git clone --submodules <url>
C. git clone <url> --with-submodules
D. git submodule clone <url>

Answer: A

Explanation: The ‘--recurse-submodules‘ flag initializes and updates submodules after cloning, ensuring dependencies are present.

57. How do you view file from specific commit?

A. git show <commit>:path/to/file
B. git cat <commit> file
C. git file show

D. git commit show file

Answer: A

Explanation: ‘git show‘ with ‘<commit>:<path>‘ prints the file contents as of that commit, useful for inspecting historical versions.

58. What does ‘git archive‘ do?

A. Create tar/zip archive of repository tree at specified commit

B. Archive commit history

C. Delete repository

D. Export tags

Answer: A

Explanation: ‘git archive‘ produces a snapshot of the repository tree, often used to package releases without Git metadata.

59. How do you list local branches sorted by last commit date?

A. git for-each-ref --sort=-committerdate refs/heads/

B. git branch --sort=-committerdate

C. Both

D. git branch --date

Answer: C

Explanation: Both commands can sort branches by commit date, with ‘for-each-ref‘ offering more formatting control.

60. What is the effect of ‘git fetch --prune‘?

A. Remove local references to deleted remote branches

B. Prune local commits

C. Delete tags

D. Clean working tree

Answer: A

Explanation: ‘--prune‘ deletes local remote-tracking branches that no longer exist on the remote, keeping references tidy.

61. Which command shows summary of changes in each commit?

A. git log --stat

B. git log --shortstat

C. Both

D. git log --summary only

Answer: C

Explanation: ‘--stat‘ displays file-level additions/deletions; ‘--shortstat‘ shows aggregated counts, offering change summaries per commit.

62. How can you create a lightweight tag?

A. git tag v1.0

B. git tag -l v1.0
C. git tag --light v1.0
D. git tag -w v1.0
Answer: A
Explanation: Omitting ‘-a‘ or ‘-m‘ creates a lightweight tag that simply references a commit without extra metadata.

63. What does ‘HEAD‘ refer to?

A. Current commit/branch reference
B. Remote repository
C. Initial commit
D. Latest tag
Answer: A
Explanation: ‘HEAD‘ points to the current branch tip or commit checked out, representing your current position in the repository.

64. How do you show commits not reachable from main?

A. git log --oneline --graph --decorate --all --not main
B. git log main..
C. Both (with proper syntax)
D. git log main only
Answer: C
Explanation: ‘git log main..‘ displays commits reachable from HEAD but not from main; ‘--not main‘ accomplishes similar filtering.

65. Which command merges and squashes branch?

A. git merge --squash feature
B. git squash feature
C. git rebase --squash feature
D. git merge feature --squash-only
Answer: A
Explanation: ‘--squash‘ combines all branch commits into one staged change, letting you commit a single squashed commit.

66. What does ‘git shortlog -sn‘ provide?

A. Summary of commits per author
B. Short commit messages
C. Stash log
D. Tag summary
Answer: A
Explanation: ‘git shortlog -sn‘ lists contributors and number of commits, useful for contribution statistics.

67. How do you compare current branch to origin/main?

A. git diff origin/main
B. git log origin/main..
C. git status -sb
D. All useful (A for diff, B commits ahead, C summary)
Answer: D
Explanation: Each command provides perspective - diff shows changes, log shows commits, and status summarises ahead/behind state.

68. Which command reapply commits on top of another branch while preserving merges?

A. git rebase --rebase-merges
B. git rebase -p (legacy)
C. Both options
D. git merge --reapply
Answer: C
Explanation: '--rebase-merges' (modern) and '-p' (legacy) preserve merge commits during rebase, maintaining branch structure.

69. How do you edit previous commit author?

A. git commit --amend --author="Name <email>"
B. git change author
C. git config author
D. git amend author
Answer: A
Explanation: Amending with '--author' overwrites author metadata for the last commit, typically used to correct identity mistakes.

70. What does 'git format-patch' generate?

A. Email-ready patch files for commits
B. Binary patch
C. Merge patch
D. Diff summary
Answer: A
Explanation: 'format-patch' outputs each commit as a patch file suitable for emailing or applying via 'git am'.

71. Which command applies patch generated by format-patch?

A. git am < patch
B. git apply patch
C. Both (git am for email patches, git apply for diffs)
D. git patch patch
Answer: C

Explanation: ‘git am‘ applies patches with metadata (committer info), while ‘git apply‘ applies raw diffs without creating commits.

72. How do you remove file from Git history but keep in working tree?

A. git rm --cached file
B. git remove file
C. git delete file
D. git clean file

Answer: A

Explanation: Removing with ‘--cached‘ untracks the file but leaves it on disk, useful for adding to ‘.gitignore‘ afterwards.

73. What is the purpose of ‘git prune‘?

A. Remove unreachable objects from object database
B. Delete branches
C. Clean working tree
D. Remove tags

Answer: A

Explanation: ‘git prune‘ garbage-collects objects not reachable from any references, typically run via ‘git gc‘.

74. Which command updates reflog expiration settings?

A. git config --global gc.reflogExpire 90.days
B. git reflog expire --expire=90.days
C. Both options possible
D. git reflog config 90

Answer: C

Explanation: Reflog expiration can be set via config or explicitly pruned with ‘git reflog expire‘, controlling how long reflog entries are kept.

75. How do you list files changed in last commit?

A. git show --name-only
B. git diff --name-only HEAD^
C. Both
D. git last --files

Answer: C

Explanation: ‘git show --name-only HEAD‘ lists files touched by the commit; ‘git diff --name-only HEAD^ HEAD‘ does the same via diff.

76. What does ‘git worktree add ../repo-feature feature‘ do?

A. Create additional working tree for branch feature
B. Add subtree
C. Create new repository

D. Add remote

Answer: A

Explanation: Git worktrees allow multiple checkouts of a repository; this command creates a new worktree at the given path for the specified branch.

77. How do you perform sparse checkout?

A. git sparse-checkout init; git sparse-checkout set <dirs>

B. git checkout --sparse

C. git clone --sparse

D. git fetch --sparse

Answer: A

Explanation: Sparse checkout uses the ‘git sparse-checkout‘ command to enable and define the directories to populate in the working tree.

78. Which command inspects Git object type?

A. git cat-file -t <object>

B. git show-type

C. git object type

D. git inspect object

Answer: A

Explanation: ‘git cat-file -t‘ prints the type (commit, tree, blob, tag) of a given object hash, aiding low-level inspection.

79. What does ‘git gc‘ perform?

A. Garbage collection to optimize repository

B. Git clone

C. General commit

D. Generate config

Answer: A

Explanation: ‘git gc‘ compresses loose objects and packs the repository, freeing space and improving performance.

80. How do you list branches containing a specific commit?

A. git branch --contains <commit>

B. git log --contains branch

C. git branch -c commit

D. git show branch commit

Answer: A

Explanation: ‘git branch --contains‘ shows branches whose history includes the commit, indicating where the change is present.

81. Which command sets default branch name for new repos?

A. git config --global init.defaultBranch main

B. git init --default-branch main
C. Both (A sets default, B per repo)
D. git default branch main
Answer: C
Explanation: You can set a global default branch name or specify it per repository during ‘git init‘.

82. What is the difference between ‘git pull --rebase‘ and ‘git pull‘?

A. Rebase applies local commits on top of fetched branch instead of merge
B. No difference
C. Pull rebase fetches only
D. Pull rebase deletes local commits
Answer: A
Explanation: ‘git pull --rebase‘ rewrites local commits atop the fetched history, avoiding merge commits and keeping linear history.

83. How do you view staged changes summary?

A. git diff --stat --cached
B. git status --stat
C. git staged --summary
D. git log --staged
Answer: A
Explanation: ‘git diff --stat --cached‘ summarizes file changes staged for commit, showing lines added/removed per file.

84. Which command clones repository with depth 1?

A. git clone --depth 1 <url>
B. git clone --shallow 1
C. git clone --depth=1
D. Both A and C
Answer: D
Explanation: ‘--depth‘ sets shallow clone depth; ‘--depth 1‘ or ‘--depth=1‘ fetch only the latest snapshot, omitting full history.

85. What does ‘git rev-parse HEAD‘ return?

A. SHA-1 or SHA-256 of current commit
B. Branch name
C. Tag
D. Tree object
Answer: A
Explanation: ‘git rev-parse‘ resolves revision strings to hashes, with ‘HEAD‘ returning the current commit hash.

86. How do you remove tracked file from repository history entirely?

A. git filter-branch --tree-filter ’rm file’
B. git filter-repo --path file --invert-paths
C. Both options depending on tools
D. git rm file
Answer: C
Explanation: History rewriting tools like ‘filter-branch‘ (legacy) or ‘filter-repo‘ remove files from commits; ‘git rm‘ only removes from tip.

87. Which command rewords commit without changing content?

A. git commit --amend --no-edit (content), or interactive rebase reword
B. git reword commit
C. git change message
D. git commit message
Answer: A
Explanation: Amending with ‘--no-edit‘ adjusts metadata; interactive rebase with ‘reword‘ edits messages while keeping changes intact.

88. What does ‘git show HEAD^‘ display?

A. Details of parent commit of HEAD
B. Next commit
C. Tree structure
D. Merge base
Answer: A
Explanation: ‘HEAD^‘ references the first parent of the current commit; ‘git show‘ reveals its details.

89. How do you create a bare repository?

A. git init --bare
B. git bare init
C. git init bare
D. git start --bare
Answer: A
Explanation: Bare repositories omit working trees, used for remotes; ‘git init --bare‘ sets up the required directory structure.

90. Which command lists branches sorted by commit count?

A. git shortlog -sn --all --no-merges
B. git for-each-ref --sort=-committerdate
C. git branch --sort=-committerdate
D. No direct built-in; use scripts
Answer: D

Explanation: Git lacks a built-in commit-count sort; scripts or ‘for-each-ref‘ with custom formatting are used to compute and sort counts.

91. What is ‘git notes‘ used for?

A. Attach additional metadata/notes to commits
B. Create commit messages
C. Manage TODOs
D. Store stash

Answer: A

Explanation: ‘git notes‘ lets you annotate commits with extra information that doesn’t modify the commit itself, stored in a separate namespace.

92. Which command configures credential helper?

A. git config credential.helper store
B. git credential store
C. git credential config
D. git config --credential store

Answer: A

Explanation: Setting ‘credential.helper‘ configures how Git caches credentials (e.g., store, cache, manager), facilitating automatic authentication.

93. How do you show only merge commits?

A. git log --merges
B. git log --only-merges
C. git log --merge-only
D. git log --type=merge

Answer: A

Explanation: ‘git log --merges‘ filters commits to merge commits, useful for reviewing integration history.

94. What does ‘git blame -L 10,20 file‘ show?

A. Line-by-line authorship for lines 10 through 20
B. Log entries
C. Commit graph
D. Merge info

Answer: A

Explanation: The ‘-L‘ option restricts the blame output to specific line ranges, focusing on relevant sections.

95. Which command copies commits from one branch to another while keeping history linear?

A. git rebase source
B. git merge source --ff-only
C. Both depending on context

D. git cherry source

Answer: C

Explanation: Rebasing replays commits onto a branch; fast-forward merges integrate commits without merge commits if possible.

96. How do you configure Git to sign tags by default?

A. git config --global tag.gpgSign true

B. git config --global tag.sign true

C. git tag --gpg default

D. git config --global gpg.tag true

Answer: A

Explanation: Setting ‘tag.gpgSign‘ ensures new tags are signed automatically when created, enforcing authenticity.

97. What does ‘git log --graph --oneline --decorate‘ help visualize?

A. Branch history in graph format

B. File status

C. Tags only

D. Stashes

Answer: A

Explanation: The combination of flags produces a compact commit graph with branch/tag names, revealing branching structure clearly.

98. Which command lists tracked files with mode and hash?

A. git ls-files --stage

B. git status --long

C. git show-files

D. git list tracked

Answer: A

Explanation: ‘git ls-files --stage‘ lists index entries with mode, object hash, and stage, useful for low-level inspection.

99. How do you abort an in-progress merge?

A. git merge --abort

B. git reset --merge

C. Both (depending on state)

D. git cancel merge

Answer: C

Explanation: During a merge, ‘git merge --abort‘ is preferred; ‘git reset --merge‘ works similarly. Use whichever fits the Git version.

100. What is the safest way to rewrite history on public branch?

A. Avoid rewriting; use new commits or revert

B. git push --force
C. git reset --hard
D. git filter-branch
Answer: A
Explanation: Rewriting shared history can disrupt collaborators; safest practice is to avoid it and use reverts or follow protocols ensuring coordination.