GitHub CLI Cheatsheet

By Dejan Panovski • Updated on Sep 23, 2026

gh at a glance: authentication, repositories, pull requests, issues, GitHub Actions runs and secrets, releases, search, extensions, and the API.

The gh command brings GitHub itself into the terminal, covering the parts of a project that live on the server rather than in the repository: pull requests, issues, workflow runs, releases, and secrets. This cheatsheet collects the subcommands and flags you reach for daily, plus the JSON output options that make gh usable inside scripts.

Authentication

Log in once per host, then check what the stored token can do.

`gh auth login`	Start the interactive login flow
`gh auth login --web`	Authenticate through the browser
`gh auth login --hostname github.example.com`	Log in to a GitHub Enterprise Server host
`gh auth status`	Show the active account, protocol, and token scopes
`gh auth refresh -s workflow`	Add a scope to the existing token
`gh auth switch`	Change the active account
`gh auth token`	Print the raw token `gh` is using
`gh auth setup-git`	Configure `git` to authenticate through `gh`
`gh auth logout`	Remove stored credentials for a host
`GH_TOKEN`	Token environment variable for GitHub.com automation
`GH_ENTERPRISE_TOKEN`	Token environment variable for GitHub Enterprise Server automation

Treat the output of `gh auth token` as a password. Store automation tokens in your CI secret manager, expose them through the appropriate environment variable, and never commit them to a repository.

Repositories

Create, clone, and inspect repositories without leaving the shell.

`gh repo clone OWNER/REPO`	Clone a repository using your configured protocol
`gh repo view`	Show the description, default branch, and README
`gh repo view --web`	Open the repository in a browser
`gh repo create NAME --private --source=. --push`	Publish the current directory as a new repository
`gh repo fork --clone`	Fork a repository and clone the fork
`gh repo sync`	Update a fork from its upstream
`gh repo list OWNER`	List repositories owned by a user or organization
`gh repo edit --default-branch main`	Set `main` as the default branch
`gh repo set-default OWNER/REPO`	Pin the repository that commands target here
`gh repo archive`	Archive a repository

Pull Requests

The commands that replace most of the pull request web interface.

`gh pr list`	List open pull requests
`gh pr list --state all --author @me`	List your pull requests in every state
`gh pr status`	Show pull requests relevant to you
`gh pr create --fill`	Open a pull request using the commit title and body
`gh pr create --draft --reviewer octocat`	Create a draft and request a review
`gh pr checkout NUMBER`	Check the pull request out as a local branch
`gh pr diff NUMBER`	Show the patch
`gh pr checks NUMBER`	Show the status checks and their conclusions
`gh pr review --approve`	Approve the pull request for the current branch
`gh pr merge NUMBER --squash`	Squash the commits and merge the pull request
`gh pr ready NUMBER`	Mark a draft as ready for review
`gh pr view NUMBER --web`	Open the pull request in a browser

Issues

Triage and close issues from the same terminal you build in.

`gh issue list`	List open issues
`gh issue list --label bug --assignee @me`	List open bugs assigned to you
`gh issue status`	Show issues assigned to or mentioning you
`gh issue create --title "..." --body "..."`	Create an issue without the prompts
`gh issue view NUMBER`	Read an issue in the terminal
`gh issue comment NUMBER --body "..."`	Add a comment
`gh issue edit NUMBER --add-label bug`	Add the `bug` label to an issue
`gh issue close NUMBER --comment "..."`	Close with a closing comment
`gh issue develop NUMBER --checkout`	Create and check out a branch linked to the issue
`gh issue transfer NUMBER OWNER/REPO`	Move an issue to another repository

GitHub Actions

Watch, debug, and re-run workflows from the command line.

`gh run list`	List recent workflow runs
`gh run list --workflow ci.yml --status failure`	List failed runs for `ci.yml`
`gh run view ID`	Show a summary of one run
`gh run view ID --log-failed`	Print the logs for failed steps only
`gh run watch ID`	Follow a run until it finishes
`gh run rerun ID --failed`	Re-run only the jobs that failed
`gh run download ID`	Download the artifacts a run produced
`gh run cancel ID`	Cancel a run in progress
`gh workflow list`	List the workflows in the repository
`gh workflow run build.yml -f env=staging`	Trigger a `workflow_dispatch` workflow with inputs
`gh workflow disable NAME`	Disable a workflow
`gh cache list`	List the Actions caches for the repository
`gh cache delete --all`	Immediately delete every Actions cache

Deleting all Actions caches has no dry-run and can slow the next workflow runs while the caches rebuild. Confirm that the command targets the intended repository before running it.

Secrets and Variables

Secrets are write-only; variables stay readable.

`gh secret set NAME`	Prompt for a value and store it, hidden from shell history
`echo "$VALUE" | gh secret set NAME`	Read the value from standard input
`gh secret set --env-file .env`	Load several secrets from a dotenv file
`gh secret set NAME --env production`	Set a deployment environment secret
`gh secret set NAME --org ORG --visibility selected --repos a,b`	Set an organization secret for named repositories
`gh secret list`	List secret names and update times, never values
`gh secret delete NAME`	Immediately remove a repository secret
`gh variable set NAME --body value`	Store a non-sensitive value that workflows can read
`gh variable get NAME`	Print a variable value
`gh variable list`	List variables with their values

Keep any dotenv file used with `--env-file` out of version control, and delete it once the values are uploaded. Before deleting a secret, confirm the repository and secret name because GitHub cannot return the stored value afterward.

Releases

Tag, publish, and fetch release assets.

`gh release create TAG --generate-notes`	Create a release with notes built from merged pull requests
`gh release create TAG --target main --title "..."`	Create the tag on a specific branch or commit
`gh release create TAG dist/app dist/checksums.txt`	Attach build files while creating the release
`gh release create TAG --draft --prerelease`	Publish as a draft or a prerelease
`gh release list`	List releases
`gh release view TAG`	Show the notes and assets for one release
`gh release download TAG`	Download every asset
`gh release download TAG --pattern "*.tar.gz"`	Download matching assets only
`gh release upload TAG file`	Add an asset to an existing release
`gh release delete TAG`	Delete a release

Search and Browse

Find work across GitHub, then jump to it.

`gh search repos QUERY --language go`	Search repositories
`gh search issues QUERY --state open`	Search issues
`gh search prs --review-requested @me`	Search pull requests
`gh search code QUERY`	Search code
`gh search commits QUERY`	Search commit messages
`gh browse`	Open the current repository in a browser
`gh browse NUMBER`	Open an issue or pull request
`gh browse path/to/file.go`	Open a file at the current branch
`gh browse --actions`	Open the Actions tab
`gh browse -n path/to/file`	Print the URL instead of opening it
`gh status`	Show issues, pull requests, and notifications across repositories

JSON Output and the API

Structured output for scripts, and direct API access for everything else.

`gh pr list --json number,title,author`	Return selected fields as JSON
`gh pr list --json number,title --jq '.[].title'`	Filter JSON without a separate `jq` process
`gh run list --json conclusion --template ...`	Format JSON with a Go template
`gh help formatting`	See every JSON formatting option
`gh pr list --json`	Passing no value lists the fields that command exposes
`gh api repos/{owner}/{repo}/releases`	Send an authenticated REST request
`gh api ENDPOINT --jq '.[].tag_name'`	Filter the API response
`gh api ENDPOINT -f title="..." -f body="..."`	Send a POST with string fields
`gh api ENDPOINT -X PATCH -F draft=false`	Choose the method and send typed fields
`gh api ENDPOINT --paginate`	Follow pagination and return every page
`gh api graphql -f query='...'`	Send a GraphQL query

The `{owner}` and `{repo}` placeholders are filled from the current repository.

Aliases, Extensions, and Config

Shorten what you type, and add commands `gh` does not ship with.

`gh alias set pv 'pr view --web'`	Create a shortcut
`gh alias set bugs 'issue list --label bug'`	Alias a command with its flags
`gh alias list`	List configured aliases
`gh alias delete NAME`	Remove an alias
`gh extension search QUERY`	Find extensions
`gh extension install OWNER/gh-NAME`	Install an extension
`gh extension list`	List installed extensions
`gh extension upgrade --all`	Update every installed extension
`gh extension remove NAME`	Remove an extension
`gh config set editor vim`	Set the editor gh opens for bodies and prompts
`gh config set git_protocol ssh`	Choose HTTPS or SSH for Git operations
`gh config list`	Show current configuration

Extensions are not reviewed or signed by GitHub, and they run with your token. Read the source before installing one.

Troubleshooting

What the common failures mean and where to look.

`To get started with GitHub CLI, please run: gh auth login`	No credentials for this host; run `gh auth login`, or set `GH_TOKEN` in CI
`HTTP 403: Resource not accessible by personal access token`	Check with `gh auth status`; refresh OAuth scopes, or fix fine-grained token access, SSO authorization, or Actions permissions
`error: your authentication token is missing required scopes [workflow]`	Refresh with the scope named in the message
`none of the git remotes configured for this repository point to a known GitHub host`	Run inside a GitHub clone, pass `--repo OWNER/REPO`, or run `gh repo set-default`
`multiple remotes detected. please select which repo to use`	Set the target once with `gh repo set-default`
`pull request create failed: GraphQL: No commits between main and branch`	Commit and push the branch before creating the pull request
`HTTP 404: Not Found on a private repository`	The account is authenticated but lacks access, or SSO is not authorized for the token
`X status with no logs in gh run view`	Logs expire with the retention policy; check the run age

Related Guides

Deeper reading on GitHub CLI and the Git commands beside it.

GitHub CLI: Manage Repositories, Issues, and Pull Requests	Full `gh` walkthrough with examples and troubleshooting
git clone Command	Clone repositories directly with Git
git diff Command	Read the patch format `gh pr diff` prints
How to Add Git Remotes	Manage the remotes `gh repo create` configures
gitignore: Ignoring Files in Git	Keep secrets and build output out of a repository
Git cheatsheet	Quick reference for the Git commands themselves