RABBIT HOLES

Accelerate · Manipal University Jaipur

13

PHASE

# The gh CLI in full

Never leave the terminal, and reach the parts of GitHub that have no command.

Part of **git started**, Accelerate's third workshop.
github.com/accelerate-muj/git-started · MIT licensed.

# Contents


1 Why bother 3

2 Auth and config 3

3 Repositories 3

4 Pull requests 4

5 Issues, releases, workflows 4

6 gh api 5

- 6.1 Methods and fields 5
- 6.2 Filtering output 5
- 6.3 GraphQL 6

7 Aliases 6

8 Extensions 6

9 gh inside Actions 7

10 Reference 8

## 1 Why bother

The web interface is fine. The argument for `gh` is not speed of a single action, it is that everything becomes scriptable.

"Add twelve people to a repository" is twelve pages of clicking, or one loop. "Which of my pull requests have failing checks" is a lot of tab switching, or one command. Phase 3's collaborator invitations were a `gh api` call for exactly this reason.

## 2 Auth and config

```
gh auth login
gh auth status
gh auth token
gh auth refresh --scopes admin:org,workflow
gh auth logout
gh auth switch
```

`gh auth refresh` is worth knowing. When a command fails with a permissions error, it is usually a missing scope rather than a real permission problem, and this adds one without logging out.

```
gh config set editor "code --wait"
gh config set git_protocol ssh
gh config set prompt disabled
gh config list
```

## 3 Repositories

```
gh repo create my-project --public --clone
gh repo create --source=. --public --push
gh repo create team-alpha --template accelerate-muj/git-started-team --public --clone
gh repo clone accelerate-muj/git-started
gh repo fork accelerate-muj/git-started --clone
gh repo sync my-username/git-started
gh repo view accelerate-muj/git-started --web
gh repo list accelerate-muj --limit 100
gh repo edit --description "..." --add-topic git --add-topic workshop
gh repo archive old-project
gh repo delete unwanted-project
```

> **NOTE**
>
> `gh repo delete` needs the `delete_repo` scope, which is not granted by default. That is deliberate friction on an irreversible action, and it is a reasonable design.

## 4 Pull requests

The largest command group, and the one that saves the most time.

```
gh pr create --fill
gh pr create --title "..." --body "..." --base main --draft
gh pr create --reviewer shashwat-deep --assignee @me --label enhancement
gh pr list --state open --author @me
gh pr list --search "review-requested:@me"
gh pr status
gh pr view 42
gh pr view 42 --comments
gh pr diff 42
gh pr checkout 42
gh pr review 42 --approve
gh pr review 42 --request-changes --body "See the comment on line 40"
gh pr review 42 --comment --body "Two small things"
gh pr merge 42 --squash --delete-branch
gh pr merge 42 --merge --auto
gh pr close 42
gh pr ready 42
gh pr checks 42
gh pr comment 42 --body "Rebased onto main"
```

Two of these deserve highlighting.

`gh pr checkout 42` fetches the contributor's branch, including from a fork, and checks it out locally. It handles the remote juggling for you. Reviewing code you can run is a different activity from reading a diff, and this is what makes it a one-command activity.

`gh pr merge --auto` queues the merge for when checks pass. Enable it and walk away, rather than watching a CI run for eleven minutes.

## 5 Issues, releases, workflows

```
gh issue create --title "..." --body "..." --label bug --assignee @me
gh issue list --state open --label bug --limit 50
gh issue develop 42 --checkout
```

`gh issue develop` creates a branch linked to an issue and checks it out. The link shows on the issue, so the connection exists before the pull request does.

```
gh release create v1.0.0 --generate-notes
gh release upload v1.0.0 ./dist/app.zip
gh release download v1.0.0 --pattern "*.zip"
gh release list
```

```
gh workflow list
gh workflow run deploy.yml --ref main -f environment=staging
gh workflow disable deploy.yml
```

```
gh run list --workflow=ci.yml --limit 10
gh run view <run-id> --log
gh run view <run-id> --log-failed
gh run watch <run-id>
gh run rerun <run-id> --failed
gh run cancel <run-id>
```

`gh run view --log-failed` prints only the failing steps' output. On a large matrix build that is the difference between reading six lines and scrolling through four thousand.

# 6 gh api

The escape hatch, and the reason `gh` can do anything GitHub can.

```
gh api user
gh api repos/accelerate-muj/git-started
gh api repos/:owner/:repo/branches
```

`:owner` and `:repo` are filled in from the current directory's repository.

## 6.1 Methods and fields

```
gh api -X PUT repos/:owner/:repo/collaborators/someone -f permission=push
gh api -X POST repos/:owner/:repo/issues -f title="Bug" -f body="Details"
gh api -X PATCH repos/:owner/:repo -F has_issues=true
gh api -X DELETE repos/:owner/:repo/collaborators/someone
```

> **TRAP**
>
> `-f` sends the value as a string. `-F` lets git interpret it, so booleans and numbers are sent as booleans and numbers.
>
> `-f has_issues=true` sends the string `"true"`, which some endpoints reject and others silently ignore. Use `-F` for anything that is not text. This is a recurring half hour of confusion for people scripting against the API.

## 6.2 Filtering output

`gh` has `jq` built in, so you do not need it installed:

```
gh api repos/:owner/:repo/pulls --jq '.[].title'
gh api repos/:owner/:repo/pulls --jq '.[] | "\(.number)  \(.user.login)  \(.title)"'
gh api repos/accelerate-muj/git-started/contributors --jq '.[].login'
gh api user/repos --paginate --jq '.[] | select(.fork == false) | .name'
```

`--paginate` follows every page automatically, which matters because the API returns thirty items by default and quietly truncates otherwise.

### 6.3 GraphQL

Some things, notably Projects, only exist in the GraphQL API:

```
gh api graphql -f query='
  query {
    viewer {
      login
      repositories(first: 5, orderBy: {field: PUSHED_AT, direction: DESC}) {
        nodes { name pushedAt }
      }
    }
  }'
```

**TRY IT**

Phase 3's collaborator loop, for a whole team at once:

```
for user in alice bob charlie dev; do
  gh api -X PUT "repos/$(gh repo view --json nameWithOwner -q .nameWithOwner)/collaborators/
    ↪ $user" -f permission=push
done
```

Then check who has accepted:

```
gh api repos/:owner/:repo/invitations --jq '.[].invitee.login'
```

## 7 Aliases

```
gh alias set prs 'pr list --author @me'
gh alias set review 'pr list --search "review-requested:@me"'
gh alias set co 'pr checkout'
gh alias set bugs 'issue list --label bug --state open'
gh alias list
```

Shell aliases with `--shell` can take arguments and use pipes:

```
gh alias set --shell teamup 'gh api -X PUT repos/$1/collaborators/$2 -f permission=push'
gh teamup accelerate-muj/git-started someone
```

## 8 Extensions

```
gh extension search
gh extension install dlvhdr/gh-dash
gh extension list
gh extension upgrade --all
```

`gh-dash` is a terminal dashboard of your pull requests and issues, and it is the one most people keep. Extensions are just repositories named `gh-something` with an executable, so writing one

is a short afternoon.

## 9 gh inside Actions

`gh` is preinstalled on GitHub-hosted runners. It needs a token in the environment:

```
- name: Comment on the PR
  env:
    GH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
  run: gh pr comment "$PR" --body "Build finished"
```

This is exactly how the Phase 2 poem checker works in this repository. Look at `.github/workflows/check-poem.yml` for a complete example.

> **TRAP**
>
> The automatic `GITHUB_TOKEN` is scoped to the repository the workflow runs in, and its permissions are whatever the `permissions:` block grants. If a `gh` call in CI fails with a 403, the fix is almost always adding the right permission to that block rather than creating a personal token.
> Reach for a personal access token only when you genuinely need to touch a *different* repository, and store it as a secret.

## 10 Reference

| Goal | Command |
| --- | --- |
| Repo from the current folder | `gh repo create --source=. --public --push` |
| Repo from a template | `gh repo create <name> --template <repo>` |
| Check out someone's PR | `gh pr checkout 42` |
| Merge when checks pass | `gh pr merge 42 --squash --auto` |
| Only the failing CI output | `gh run view <id> --log-failed` |
| Branch linked to an issue | `gh issue develop 42 --checkout` |
| Add a collaborator | `gh api -X PUT repos/:owner/:repo/collaborators/<u>` |
| Every page of a list | `gh api <endpoint> --paginate` |
| Boolean field to the API | `-F key=true`, never `-f` |
| Save a command as a word | `gh alias set <name> '<command>'` |

**CHECK YOURSELF**

1. What does `gh pr checkout` do that `git fetch` alone does not?
2. Difference between `-f` and `-F` on `gh api`, and when it bites.
3. Why do you need `--paginate`?
4. A `gh` call in a workflow returns 403. What is the first thing to check?
5. Write an alias that lists pull requests waiting on your review.

**Next:** Phase 14, *Actions and CI/CD*, on making the repository do work by itself: running tests on every pull request, and deploying when you tag a release.