Git + GitHub Actions Cheat Sheet

A quick reference for the commands and workflow syntax you reach for every day. Bookmark it, print it, or keep it in your notes.

Part 1 — Git: the commands that matter

Undo things (the reason most people search)

You want to	Command	Notes
Discard unstaged changes in a file	`git restore <file>`	Modern form of `git checkout -- <file>`
Unstage a file (keep the changes)	`git restore --staged <file>`	Modern form of `git reset HEAD <file>`
Amend the last commit message	`git commit --amend`	Only if not pushed
Add a forgotten file to the last commit	`git add <file> && git commit --amend --no-edit`	Only if not pushed
Undo the last commit, keep the changes staged	`git reset --soft HEAD~1`	
Undo the last commit, keep the changes unstaged	`git reset HEAD~1`	Default (`--mixed`)
Throw away the last commit entirely	`git reset --hard HEAD~1`	Destructive — the work is gone
Revert a pushed commit safely	`git revert <sha>`	Creates a new commit that undoes it
Recover a "lost" commit	`git reflog` then `git checkout <sha>`	Reflog keeps ~90 days of HEAD history

Branching

```
git switch -c feature/x       # create + switch (like git checkout -b)
git switch main               # switch to an existing branch
git branch -d feature/x       # delete a merged branch
git branch -D feature/x       # force-delete an unmerged branch
git push -u origin feature/x  # push + set upstream (first push)
git fetch --prune             # prune refs for remote-deleted branches
```

Merge conflicts

```
git merge main                  # or: git rebase main
# ... conflict ...
git status                      # lists the conflicted files
# edit files, remove <<<<<<< ======= >>>>>>> markers
git add <resolved-file>
git merge --continue            # or: git rebase --continue
git merge --abort               # bail out, back to pre-merge state
```

Set a merge tool once: `git config --global merge.tool <vimdiff|vscode|meld>`

Rebase vs merge — which to use

- **Merge** into shared/long-lived branches (`main`, `develop`). Preserves history as it happened.
- **Rebase** your own feature branch onto `main` before opening a PR — linear history, easier review.
- **Never** rebase a branch other people have pulled. Rewriting shared history breaks their clones.
- `git pull --rebase` instead of plain `git pull` avoids noise merge commits on every sync.

Stash

```
git stash push -m "wip: auth refactor"
git stash list
git stash pop                   # apply + drop the latest
git stash apply stash@{2}       # apply a specific one, keep it
git stash drop stash@{2}
```

Tags & releases

```
git tag -a v1.4.0 -m "Release 1.4.0"   # annotated — use for releases
git push origin v1.4.0
git push origin --tags                 # push all tags
git tag -d v1.4.0                      # delete the tag locally
git push origin :refs/tags/v1.4.0      # delete it on the remote
```

Cherry-pick & clean

```
git cherry-pick <sha>          # apply one commit onto HEAD
git cherry-pick <sha1>^..<sha2>  # apply a range (incl. sha1)
git cherry-pick --abort        # bail out of a conflicted pick
git clean -n                   # preview what would be deleted
git clean -fd                  # delete untracked files + dirs
git clean -fdx                 # ...also .gitignored files
```

Inspect

```
git log --oneline --graph --decorate --all
git log -p <file>              # history of one file with diffs
git blame <file>
git show <sha>
git diff main...feature/x      # what the branch adds vs main
```

Part 2 — GitHub Actions: workflow syntax

Minimal workflow

```yaml
# .github/workflows/ci.yml
name: CI
on:
  push:
    branches: [main]
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v5
      - uses: actions/setup-node@v5
        with:
          node-version: 24        # Active LTS in 2026
          cache: npm
      - run: npm ci
      - run: npm test
```

> **Action versions:** examples pin `@v5`; the latest majors are `checkout@v7`, `setup-node@v7`, `cache@v4` — check each action's release notes before bumping, since `checkout` v6/v7 changed `pull_request_target` fork-checkout behavior.

Trigger reference (`on:`)

Trigger	Fires when
`push`	Commits pushed (filter with `branches`, `paths`, `tags`)
`pull_request`	PR opened/synchronized/reopened against the repo
`pull_request_target`	Same, but runs in the **base** repo context — has secrets. Dangerous with untrusted code.
`workflow_dispatch`	Manual "Run workflow" button (define `inputs`)
`schedule`	Cron, e.g. `- cron: '0 6 * * 1'` (UTC only)
`workflow_call`	Called by another workflow (reusable workflows)
`release`	A release is published/created

Path & branch filters

```yaml
on:
  push:
    branches: ['main', 'release/**']
    paths: ['src/**', '!**.md']
```

Permissions (least privilege — set this)

```
permissions:
  contents: read          # default to read-only
# raise per-job only where needed, e.g.:
#   contents: write       # to push tags/commits
#   packages: write       # to publish to GHCR
#   id-token: write       # for OIDC cloud auth
```

Secrets & variables

```
steps:
  - run: ./deploy.sh
    env:
      API_TOKEN: ${{ secrets.API_TOKEN }}
      REGION: ${{ vars.AWS_REGION }}
```

- Secrets are masked in logs; variables are not.
- `GITHUB_TOKEN` is auto-provided per run — scope it with `permissions:`, don't create a PAT.
- Prefer **OIDC** (`id-token: write` + cloud trust policy) over long-lived cloud keys in secrets.

Matrix builds

```
strategy:
  fail-fast: false
  matrix:
    os: [ubuntu-latest, macos-latest]
    node: [22, 24, 26]
runs-on: ${{ matrix.os }}
```

Caching

```
- uses: actions/cache@v4
  with:
    path: ~/.cache/pip
    key: ${{ runner.os }}-pip-${{ hashFiles('requirements.txt') }}
    restore-keys: ${{ runner.os }}-pip-
```

Note: caches evict after 7 days unused, and the repo cache is capped at 10 GB (LRU).

Concurrency (cancel superseded runs)

```
concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true
```

Reusable workflow

```
# caller
jobs:
  deploy:
    uses: my-org/shared-workflows/.github/workflows/deploy.yml@main
    with: { environment: prod }
    secrets: inherit
```

Common expressions & contexts

Expression	Value
`${{ github.sha }}`	Commit SHA
`${{ github.ref_name }}`	Branch or tag name
`${{ github.event_name }}`	The trigger (`push` , `pull_request` , …)
`${{ github.actor }}`	User who triggered the run
`${{ runner.os }}`	`Linux` / `macOS` / `Windows`
`${{ job.status }}`	`success` / `failure` / `cancelled`
`if: github.ref == 'refs/heads/main'`	Step/job condition
`if: always()`	Run even if a previous step failed

Security quick rules

1. Pin third-party actions to a **full commit SHA**, not a tag (`uses: foo/bar@a1b2c3...`) — tags can be silently repointed.
2. Default `permissions: contents: read` ; escalate per-job only.
3. Never use `pull_request_target` to check out and run untrusted PR code.
4. Use OIDC for cloud auth instead of storing static keys.
5. Treat `${{ github.event.* }}` values (PR titles, branch names) as untrusted input — don't interpolate them straight into `run:` shell.

> Why rule 1: in March 2025 an attacker repointed every `tj-actions/changed-files` tag (v1–v45.0.7) at a malicious commit that dumped CI secrets into build logs across ~23,000 repos (CVE-2025-30066). A pinned SHA can't be moved out from under you.