



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Chapter 02

REMOTE-TRACKING BRANCHES

The Problem: Ghost Branches

After merging PRs, you might see branches that don't exist anymore:

```
$ git branch -a  
main  
remotes/origin/main  
remotes/origin/feature/old-pr ← This was deleted on GitHub!
```

Why does this happen?

Git doesn't automatically clean up references to deleted remote branches.

The Three Types of Branches

When you run `git branch -a`, you see **three different things**:

```
$ git branch -a
main                ← Local branch
feature/new-feature ← Local branch
remotes/origin/main  ← Remote-tracking branch
remotes/origin/feature/xyz ← Remote-tracking branch
```

Let's understand each type...

Type 1: Local Branches

```
main  
feature/new-feature
```

Local branches:

- Your own branches where you work
- Exist only on your computer
- You create them with `git checkout -b`
- You control when they're created and deleted

Type 2: Remote-Tracking Branches

```
remotes/origin/main  
remotes/origin/feature/xyz
```

Remote-tracking branches:

- **Local copies** of what exists on GitHub
- Automatically updated when you `git fetch` or `git pull`
- You don't directly work on these—they're bookmarks showing GitHub's state
- Named with `remotes/origin/` prefix

Type 3: The Remote Repository (GitHub)

The actual branches on GitHub:

- The source of truth
- What your team sees
- Where PRs are merged
- Where branches get deleted after merging

The Bookmark Analogy

Think of remote-tracking branches like **bookmarks in your browser**:

Git Concept	Browser Analogy
GitHub	The actual websites
Remote-tracking branches	Your bookmarks pointing to GitHub's branches
Local branches	Your own notes/drafts

The problem: When a branch is deleted on GitHub (after merging a PR), your bookmark still points to it!

Why Stale References Happen

Typical workflow:

1. ✓ You create a feature branch and push it to GitHub
2. ✓ You create a PR and it gets merged
3. ✓ GitHub **automatically deletes** the branch after merge
4. ✗ Your local Git **still has** `remotes/origin/feature/xyz` (stale!)

Git doesn't automatically remove these references because:

- It doesn't know if the deletion was intentional
- You might have been offline when the deletion happened
- Git errs on the side of keeping information

Example: Stale References in Action

After PR is merged and branch deleted on GitHub:

```
$ git branch -a  
main  
remotes/origin/main  
remotes/origin/feature/old-pr ← Stale! No longer exists on GitHub
```

This causes confusion:

- `git branch -a` shows branches that don't exist
- You might accidentally try to work on deleted branches
- Repository looks cluttered

Solution 1: `git fetch --prune`

The `--prune` flag tells Git: "Remove local references to remote branches that no longer exist."

```
$ git fetch --prune
```

What this does:

1. **Fetch**: Download new commits and branches from GitHub
2. **Prune**: Remove remote-tracking branches that no longer exist on GitHub

`git fetch --prune` Example

Before pruning:

```
$ git branch -a  
main  
remotes/origin/main  
remotes/origin/feature/old-pr  
remotes/origin/feature/another-merged
```

Running prune:

```
$ git fetch --prune  
From https://github.com/user/repo  
- [deleted]          (none)      -> origin/feature/old-pr  
- [deleted]          (none)      -> origin/feature/another-merged
```

Translation: "These remote-tracking branches were deleted locally because they no longer exist on GitHub."

Solution 2: `git remote prune origin`

Alternative command:

```
$ git remote prune origin
```

Difference from `git fetch --prune`:

- `git fetch --prune`: Downloads new data **AND** removes stale references (recommended)
- `git remote prune origin`: **Only** removes stale references (doesn't fetch new data)

Preview Before Pruning

See what would be removed without actually removing:

```
$ git remote prune origin --dry-run
Pruning origin
URL: https://github.com/user/repo
* [would prune] origin/feature/old-pr
* [would prune] origin/feature/another-merged
```

Use this when you're not sure what will be deleted!

Complete Branch Cleanup Workflow

After merging a PR and having the branch deleted on GitHub:

```
# Step 1: Switch to main and update it (with automatic pruning)
git checkout main
git pull --prune

# Step 2: Delete your local feature branch
git branch -d feature/my-merged-feature
```

Understanding `git pull --prune`

`git pull --prune` does **THREE** things at once:

1. **Fetches** updates from GitHub (updates `origin/main` bookmark)
2. **Merges** those updates into your local main (moves main pointer forward)
3. **Removes** stale remote-tracking branches (like `remotes/origin/feature/my-merged-feature`)

It's the most efficient cleanup command!

Visual: Before Cleanup

Your Local Git:

└─ Local branches:

| └─ main

| └─ feature/old-pr (merged, but you forgot to delete)

└─ Remote-tracking branches:

└─ remotes/origin/main ✓ (exists on GitHub)

└─ remotes/origin/feature/old-pr ✗ (deleted on GitHub)

└─ remotes/origin/feature/another-old ✗ (deleted on GitHub)

Visual: After Cleanup

Your Local Git:

|— Local branches:

| |— main (clean!)

|— Remote-tracking branches:

 |— remotes/origin/main ☒ (exists on GitHub)

Much cleaner!

Automatic Pruning (Recommended Setup)

Configure Git to always prune when fetching:

```
$ git config --global fetch.prune true
```

Now every `git fetch` and `git pull` will automatically clean up stale references!

Benefits:

- One less thing to remember
- Always have a clean view of branches
- No stale references cluttering your `git branch -a` output

Check If Auto-Prune Is Enabled

```
$ git config --global --get fetch.prune  
true
```

If it returns **true**, you're all set!

If it returns nothing, enable it:

```
$ git config --global fetch.prune true
```

What About Local Branches?

Important: Remote pruning only affects **remote-tracking branches** (`remotes/origin/*`), not your **local branches**.

You must manually delete local branches:

```
$ git branch -d feature/my-merged-feature # Delete if merged  
$ git branch -D feature/my-merged-feature # Force delete
```

Why Git keeps local branches:

- You might have uncommitted work
- You might want to reference the branch later
- It's safer to require explicit deletion

Common Scenario 1: "Not Fully Merged" Error

```
$ git branch -d feature/xyz  
error: The branch 'feature/xyz' is not fully merged.  
hint: If you are sure you want to delete it, run 'git branch -D feature/xyz'.
```

Why this happens:

PR was merged with "Squash and merge" or "Rebase and merge"—this creates new commits instead of using your original commits.

Solution: Use capital `-D` to force delete:

```
$ git branch -D feature/xyz
```

! Check first that the PR was indeed merged on GitHub before force deleting!

Common Scenario 2: Which Branches Are Merged?

Show branches merged into current branch:

```
$ git branch --merged
```

Show branches merged into main:

```
$ git branch --merged main
```

Show branches NOT yet merged:

```
$ git branch --no-merged
```

Common Scenario 3: Batch Cleanup

Delete all local branches that have been merged into main:

```
$ git branch --merged main | grep -v "^\"* main\" | xargs git branch -d
```

⚠ Warning: Only run this if you're sure all listed branches are actually merged!

Better approach: Delete branches one by one to be safe.

Why Clean Up Matters

Benefits of keeping your branches clean:

-  **Clarity:** `git branch -a` only shows branches that actually exist
-  **Faster:** Less clutter means easier to find what you need
-  **Professional:** Shows you understand Git's architecture
-  **Prevents mistakes:** Won't accidentally work on deleted branches

It's not critical: Stale references don't break anything—they just cause confusion.

Understanding the Three-Level Sync

```
GitHub (source of truth)
  ↓ git fetch/pull
Remote-tracking branches (local bookmarks)
  ↓ git merge/checkout
Local branches (your work)
```

Each level needs separate cleanup:

- GitHub: Branches deleted automatically after PR merge
- Remote-tracking: Use `git fetch --prune` or `git pull --prune`
- Local: Use `git branch -d` manually

Quick Reference: Cleanup Commands

```
# See all branches (including remote-tracking)
git branch -a
```

```
# Preview what would be pruned
git remote prune origin --dry-run
```

```
# Remove stale remote-tracking branches
git fetch --prune
```

```
# Or just prune without fetching
git remote prune origin
```

```
# Delete merged local branch
git branch -d feature/merged-branch
```

```
# Force delete local branch (if squash merged)
git branch -D feature/merged-branch
```

```
# Enable automatic pruning (recommended)
git config --global fetch.prune true
```

The Complete Post-PR Workflow

After your PR is merged on GitHub:

```
# 1. Switch to main
git checkout main

# 2. Pull latest changes and clean up remote-tracking branches
git pull --prune

# 3. Delete your local feature branch
git branch -d feature/my-merged-feature
# (or git branch -D if squash merged)

# 4. Verify cleanup
git branch -a
```

Clean repository, ready for the next feature!

Key Takeaways

1. **Remote-tracking branches** are local bookmarks to GitHub's branches
2. **Stale references** happen when branches are deleted on GitHub
3. `git fetch --prune` removes these stale references
4. **Local branches** must be deleted separately with `git branch -d`
5. **Enable automatic pruning** with `git config --global fetch.prune true`
6. **Cleanup is optional** but keeps your repository tidy

Think of branch cleanup as tidying your desk—your work doesn't depend on it, but it makes everything clearer!

Practice Exercise

Try this workflow end-to-end:

1. Create a feature branch and make a commit
2. Push to GitHub and create a PR
3. Merge the PR (GitHub deletes the branch)
4. Run `git branch -a` (see the stale reference)
5. Run `git fetch --prune` (see it removed)
6. Delete local branch with `git branch -d`
7. Enable auto-prune: `git config --global fetch.prune true`

You'll never have stale references again!

Common Questions

Q: Will pruning delete my work?

A: No! It only removes bookmarks to branches that don't exist on GitHub.

Q: What if I prune by mistake?

A: Nothing bad happens—you're just removing outdated bookmarks.

Q: Should I always enable auto-prune?

A: Yes! It's a best practice and has no downsides.

Further Reading

Official Git documentation

- [Git Branching - Remote Branches](#)
- [git-fetch documentation](#)

GitHub documentation

- [About pull requests](#)
- [GitHub flow](#)

Interactive learning

- [Learn Git Branching](#)

Pro tip: After every merged PR, run `git pull --prune` and clean up local branches!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU