Workshop: Introduction to Git and GitHub

Part 3: Git Branching

Philippe Joly

Freie Universität Berlin

March 9, 2021

Reference

- This workshop draws extensively on Scott Chacon and Ben Straub (2021), *ProGit*, Version 2.1.295, 2021-02-26.
- Like the book, this workshop carries the CC BY-NC-SA 3.0 license.

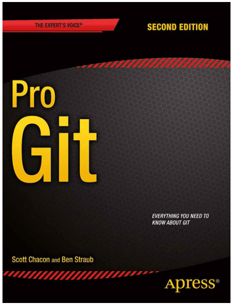
THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®

Figure 1

How Git branching works

Git branching

- A divergence from the main line of development
- Git “killer feature”
  - Lightweight
  - Fast
  - Encourages workflows that branch and merge often
  - Let’s you freely experiment
  - Structures collaboration

A branch is simply a pointer

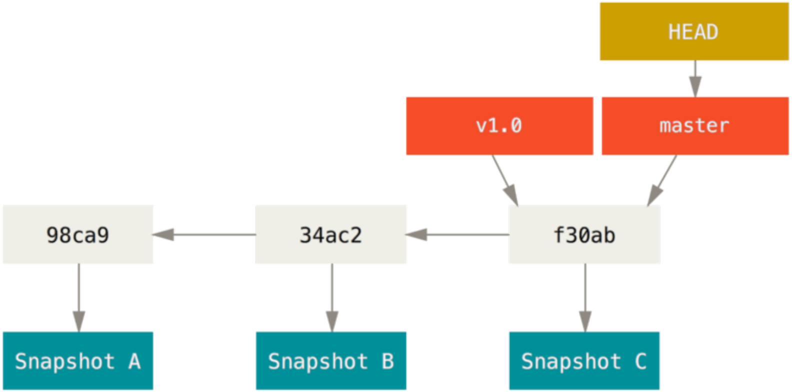

Figure 2: A branch and its commit history. *Source*: Chacon & Straub (2021), fig. 11.

Creating a branch adds a pointer to your commit history

```
$ git branch testing
```

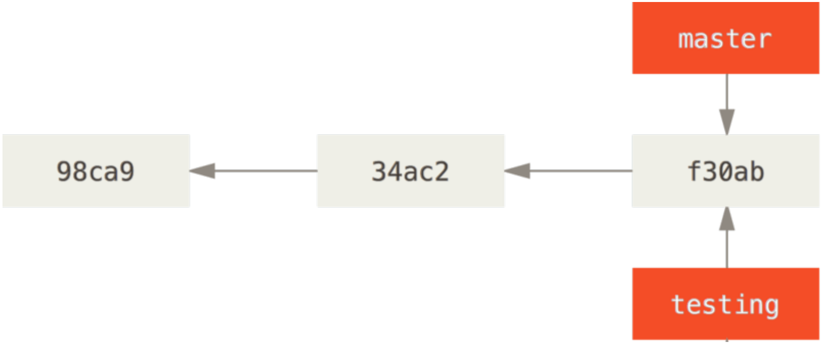

Figure 3: Two branches pointing into the same series of commits. *Source*: Chacon & Straub (2021), fig. 12.

HEAD points to your current position

```
$ git branch
* master
  testing
```

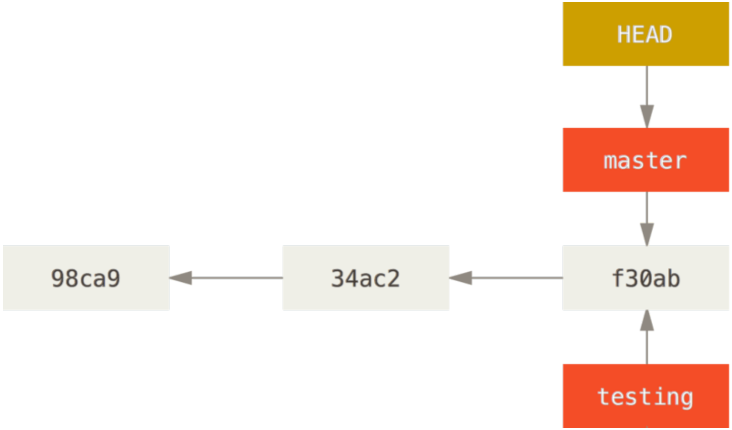

Figure 4: HEAD pointing to a branch. *Source*: Chacon & Straub (2021), fig. 13.

Switching branches

```
$ git checkout testing
$ git branch
  master
* testing
```

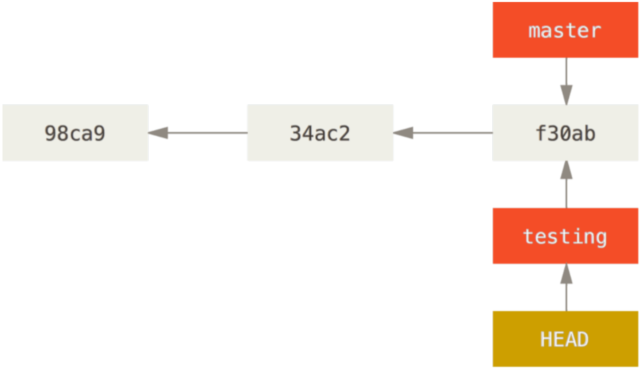

Figure 5: HEAD points to the current branch. *Source*: Chacon & Straub (2021), fig. 14.

The new branch moves forward

```
$ git add myfile.txt
$ git commit -m "add this new file"
```

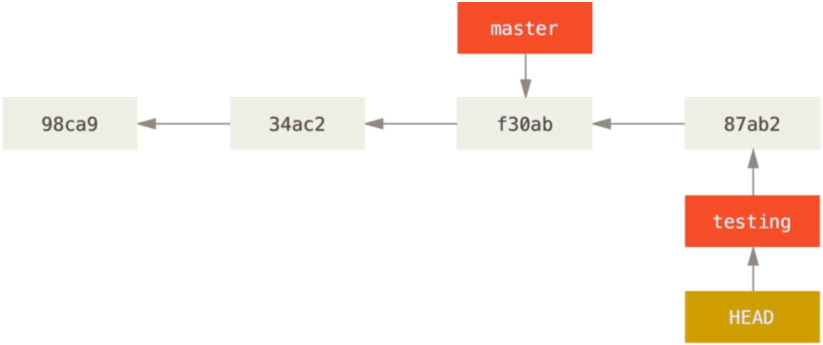

Figure 6: HEAD points to the current branch. *Source*: Chacon & Straub (2021), fig. 15.

Back to master (the main branch)

```
$ git checkout master
```

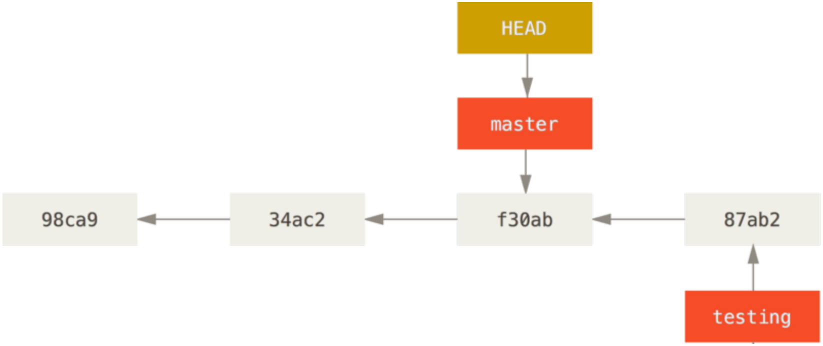

Figure 7: HEAD moves when you checkout. *Source*: Chacon & Straub (2021), fig. 16.

master moves forward: a divergent history

```
$ git add myfile2.txt
$ git commit -m "add another file"
```

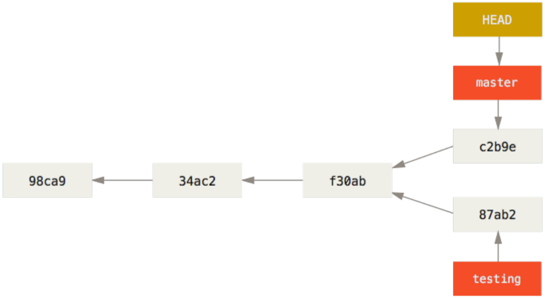

Figure 8: Divergent history. *Source*: Chacon & Straub (2021), fig. 17.

Creating and switching branches: summary

Create a new branch

```
$ git branch newbranch
```

Switch to that branch

```
$ git checkout newbranch
```

Shortcut: create *and* switch to a new branch

```
$ git checkout -b newbranch
```

List your branches and see on which one you are now

```
$ git branch
```

Merging

Basic merging (1)

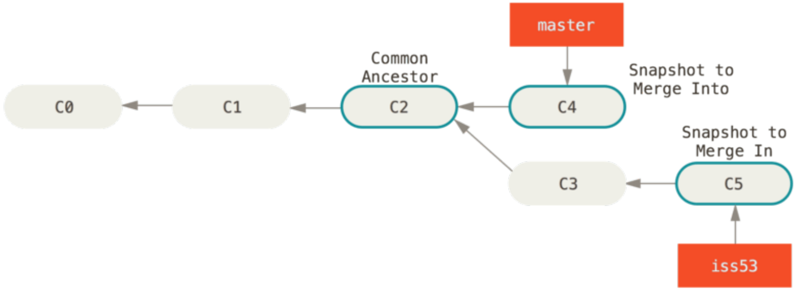

Figure 9: Three snapshots used in a typical merge. *Source*: Chacon & Straub (2021), fig. 24.

Basic merging (2)

Move (`checkout`) to the **receiving** branch (master) before merging

```
$ git checkout master
$ git merge iss53
```

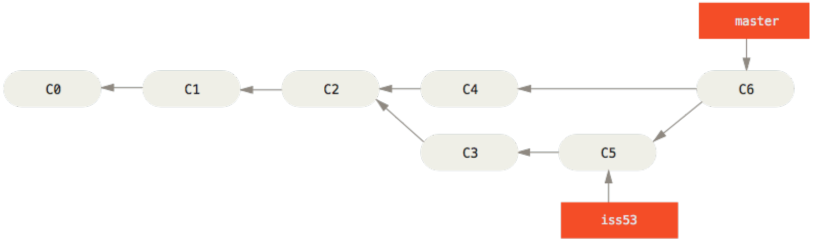

Figure 10: A merge commit. *Source*: Chacon & Straub (2021), fig. 25.

Deleting a branch

After merging, you can safely delete the branch.

```
$ git branch -d iss53
```

Merge Conflicts

Basic Merge Conflicts (1)

If you have modified the same lines, of the same file, in both branches, you will have a merge conflict.

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Basic Merge Conflicts (2)

Git handles this problem by inserting markers in your file to highlight the merge conflict.

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Basic Merge Conflicts (3)

Workflow:

1. Open the problematic file
2. Look for the <<<<<<<, =======, and >>>>>>> markers
3. Revise this part of the file: select one of the two options or create a new one.
4. Delete the markers.
5. Save your file.
6. Stage your changes using `git add`.
7. Finalize your merge using `git commit`.

Inspect your merging history

```
$ git log --oneline --graph --all

*   c545382 (HEAD -> main) Manage the merge conflict
|\
| * a6afe91 Modify the same lines on newbranch
* | e621ec9 Change something on main
|/
* 764a766 Initial commit
```

Exercise

Exercise 1 (a)

1. Open a new folder and initialize a new repo.

```
$ git init
```

2. Create a file named `list.md` with a list of **three places** you would like to visit after the pandemic.

3. Stage and commit your changes.

```
$ git add list.md
$ git commit list.md
```

Exercise 1 (b)

4. Create and switch to a new branch

```
$ git checkout -b newbranch
```

5. Modify the third item on your list.

6. Stage and commit your changes.

```
$ git add list.md
$ git commit list.md
```

7. Switch back to the main branch

```
$ git checkout main
```

8. Repeat steps 5 and 6 but modify the third item differently this time.

Exercise 1 (c)

9. Try merging

```
$ git merge newbranch
```

10. Open `list.md` and manage the merge conflict

11. Stage and finalize your merge

```
$ git add list.md
$ git commit list.md
```

12. Inspect your merging history

```
$ git log --oneline --graph --all
```

13. Delete `newbranch`

```
$ git branch -d newbranch
```

Workflows

Git Feature Branch Workflow

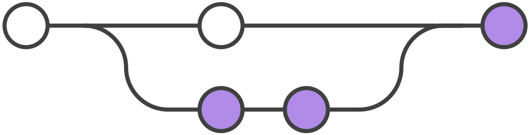

Figure 11: Feature Branch Workflow. *License*: CC BY 2.5 AU. *Source*: Atlassian. https://www.atlassian.com

- All feature development takes place in a dedicated branch.
- The main branch should not contain broken code.
- Merges are the focal point of discussion in a team.

Gitflow for more complex projects

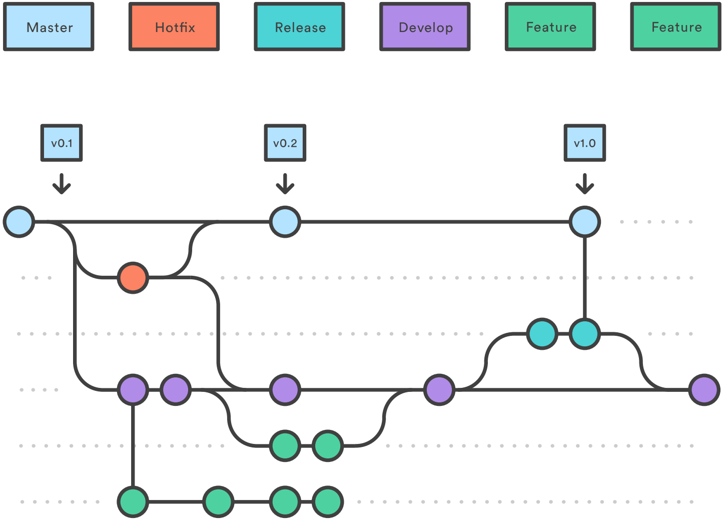

Figure 12: Gitflow. *License*: CC BY 2.5 AU. *Source*: Atlassian. https://www.atlassian.com

Refresher: working with remotes

Refresher: working with remotes (1)

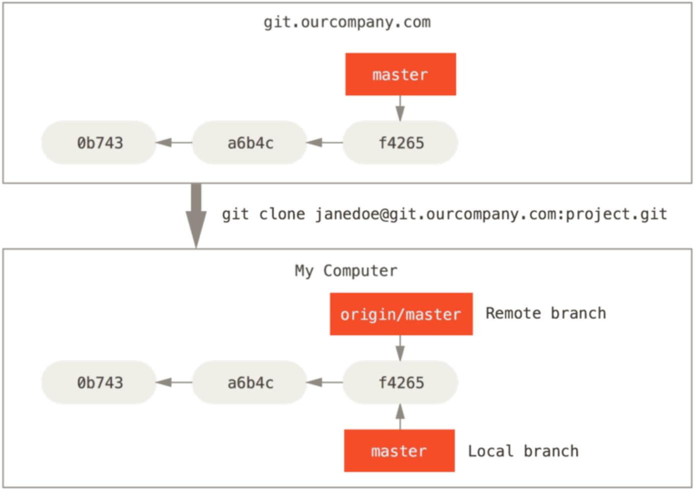

Figure 13: Server and local repositories after cloning. *Source*: Atlassian. Chacon & Straub, fig. 30.

Refresher: working with remotes (2)

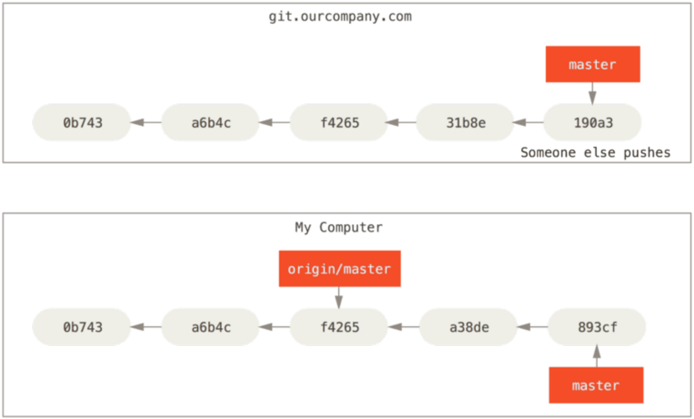

Figure 14: Local and remote work can diverge. *Source*: Atlassian. Chacon & Straub, fig. 31.

Refresher: working with remotes (3)

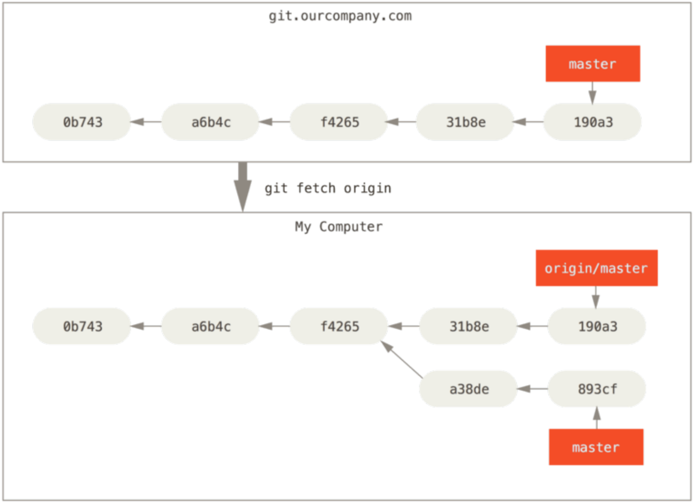

Figure 15: `git fetch` updates your remote-tracking branches. *Source*: Atlassian. Chacon & Straub, fig. 32.

Refresher: working with remotes (4)

Update the version history of the server on your computer:

```
$ git fetch origin
```

Examine the differences between your local branch and the branch on the server:

```
$ git diff origin/main
```

If needed, merge:

```
$ git merge origin/main
```

`git pull`

- `git pull` is a **shortcut** for `git fetch` and `git merge`.
- This command is dangerous because it can overwrite your working directory without giving you the chance to examine the changes.
- It's usually better to run `git fetch`, examine the changes with `git diff`, and then run `git merge`.

Exercise

Exercise 2

1. Log into GitHub and open the demo repository you created in Exercise 3 of Part 2 of the workshop.

2. Modify `README.md` and commit your changes **online**.

3. In your **local** repo, update the version history of the server.

```
$ git fetch origin
```

4. Examine the differences between your local version history and the version history on the server.

```
$ git status
$ git diff origin/main
```

5. Merge the version history of the server into your local main branch.

```
$ git merge origin/main
```

Remote branches

Working with remote branches (1)

Scenario 1: Create a new branch and share it on a remote

Create and switch to a new branch:

```
$ git checkout -b newbranch
```

Push this branch to the remote:

```
$ git push origin newbranch
```

Start tracking the remote branch:

```
$ git branch -u origin/newbranch
```

In other words, when you `git push` and `git fetch`, Git will automatically compare your local **tracking branch** with the **remote upstream** branch.

Working with remote branches (2)

Inspect your tracking and upstream branches

```
$ git branch -vv
  main      d2abadb [origin/main] Update README.md
* newbranch d2abadb [origin/newbranch] Update README.md
```

- The local `main` branch is tracking the upstream `main` branch on the remote `origin`.
- The local `newbranch` is tracking the upstream `newbranch` on the remote `origin`.

Working with remote branches (3)

Scenario 2: Import a branch created by someone else on the remote

Fetch the remote repository. Git informs you about the new branch:

```
$ git fetch
From github.com:jolyphil/demorepo
 * [new branch]    newbranch      -> origin/newbranch
```

`newbranch` is still not a local branch:

```
$ git branch
* main
```

Create a local copy and switch to it:

```
$ git checkout --track origin/newbranch
Branch 'newbranch' set up to track remote branch 'newbranch'
    from 'origin'.
Switched to a new branch 'newbranch'
```

Deleting a remote branch

Switch back to `main`:

```
$ git checkout main
```

Delete the local branch:

```
$ git branch -d mybranch
```

Delete the remote branch:

```
$ git push origin --delete mybranch
```

Exercise

Exercise 3

1. Open the repo used in the previous exercise.
2. Create and switch to a new branch named 'newfeature'.

```
$ git checkout -b newfeature
```

3. Push this branch to the remote.

```
$ git push origin newfeature
```

4. Start tracking the remote branch.

```
$ git branch -u origin/newfeature
```

5. Inspect your tracking and upstream branches.

```
$ git branch -vv
```