Workshop: Introduction to Git and GitHub

Part 2: Git Basics

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.

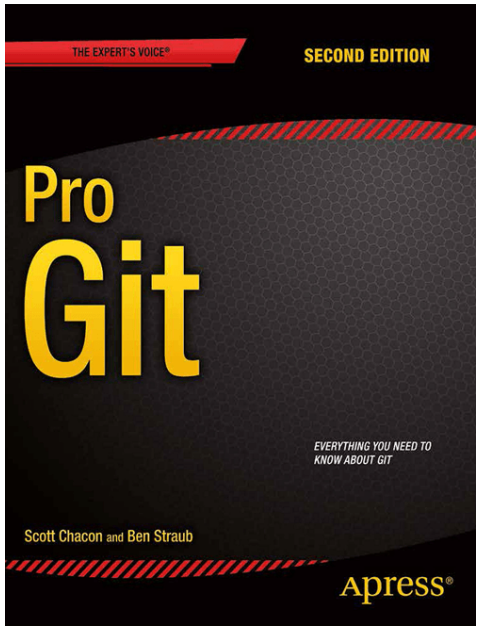

Figure 1

Two ways of getting a Git repository

1. Take a local directory that is currently not under version control and turn it into a Git repository
2. `clone` an existing Git repository from elsewhere.

Initializing a git repository (1)

- Open a project folder
- Open your terminal from inside the project folder
  - Linux (Ubuntu): right-click + open terminal here
  - Windows: right-click + Git Bash here
  - MacOS: you need to activate this functionality

Initializing a git repository (2)

You can otherwise navigate with the `cd` command.

For Linux:

```
$ cd /home/user/my_project
```

For MacOS:

```
$ cd /Users/user/my_project
```

For Windows:

```
$ cd C:/Users/user/my_project
```

Initializing a git repository (3)

Now type:

```
$ git init
```

- Creates a subdirectory named `.git`
- Might be hidden depending on your configuration
- Your project folder is now a git repository!

Staging and committing a file

Staging a file (taking a snapshot)

```
$ git add myfile.txt
```

Committing a file (saving that snapshot)

```
$ git commit myfile.txt -m "Record this change in project"
```

Cloning an existing repository

Cloning creates a copy of a repository stored on a remote server and imports the entire version history of the project.

https protocol:

```
$ git clone https://github.com/jolyphil/git-workshop
```

If you have set up an SSH connection on the server (here: GitHub), you can also clone the repo like this:

```
$ git clone git@github.com:jolyphil/git-workshop
```

Advantage: you don't have to re-enter your username and password.

Tip: `git clone` is usually easier than `git init`

- `git clone` automatically sets up the connection between your local and remote repository.
- You would usually proceed as follows:
  1. Create a new repository containing only a README file on GitHub
  2. Clone this repository on your local machine
  3. Create or copy your project files into the cloned repository
  4. Start committing

Checking the status of your files (1)

```
$ git status
On branch main
Your branch is up-to-date with 'origin/main'.
nothing to commit, working tree clean
```

Checking the status of your files (2)

What happens when you create or copy a file in your repo

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README.md

nothing added to commit but untracked files present (use "git
    add" to track)
```

Untracked files

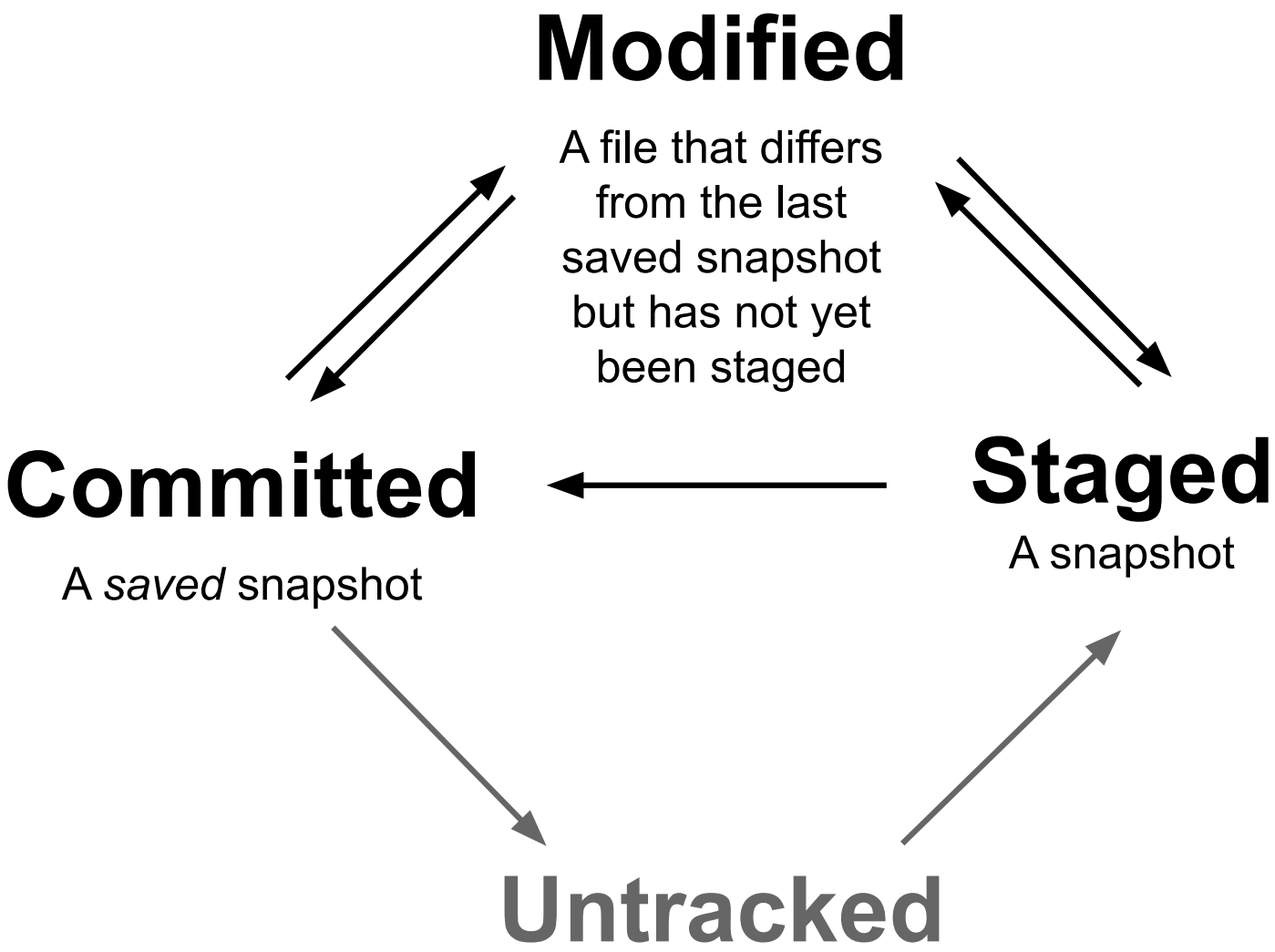

Figure 2: New files have to be explicitly added to the version history.

Tracking a new file

Add a new file

```
$ git add README.md
```

Check the status

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    new file:  README.md
```

Staging modified files (1)

What happens when you modify a tracked file (here: `CONTRIBUTING.md`)

```
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:  README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
      directory)

    modified:  CONTRIBUTING.md
```

Staging modified files (2)

After adding the modified file to the staging area

```
$ git add CONTRIBUTING.md
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:  README.md
    modified:  CONTRIBUTING.md
```

Staging modified files (3)

What happens if you modify the file again before committing

```
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:  README.md
    modified:  CONTRIBUTING.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
      directory)

    modified:  CONTRIBUTING.md
```

Staging modified files (4)

Adding the file again to the staging area.

```
$ git add CONTRIBUTING.md
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:  README.md
    modified:  CONTRIBUTING.md
```

Ignoring files

You can ignore certain files by creating a `.gitignore` file in your repo.

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not
    subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
```

Viewing Your Staged and Unstaged Changes (1)

Let's say you have a modified file named `CONTRIBUTING.md`

```
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:  README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
      directory)

    modified:  CONTRIBUTING.md
```

Viewing Your Staged and Unstaged Changes (2)

Compares your working directory with the staged area (or with the last commit if you haven't staged anything since then.)

```
$ git diff
```

If you want to compare what is currently staged with the last commit, you can proceed as follows:

```
$ git diff --staged
```

This describes what will go into your next commit.

Committing

This will open your default editor and let you write a commit message.

```
$ git commit
```

It's easier to you use the `-m` option and do everything in a single command.

```
$ git commit -m "Story 182: fix benchmarks for speed"
[main 463dc4f] Story 182: fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README.md
```

Good commit messages are important

- Short, approx. 50 characters
- Use the imperative (e.g., "Fix this problem")

What happens when you commit (1)

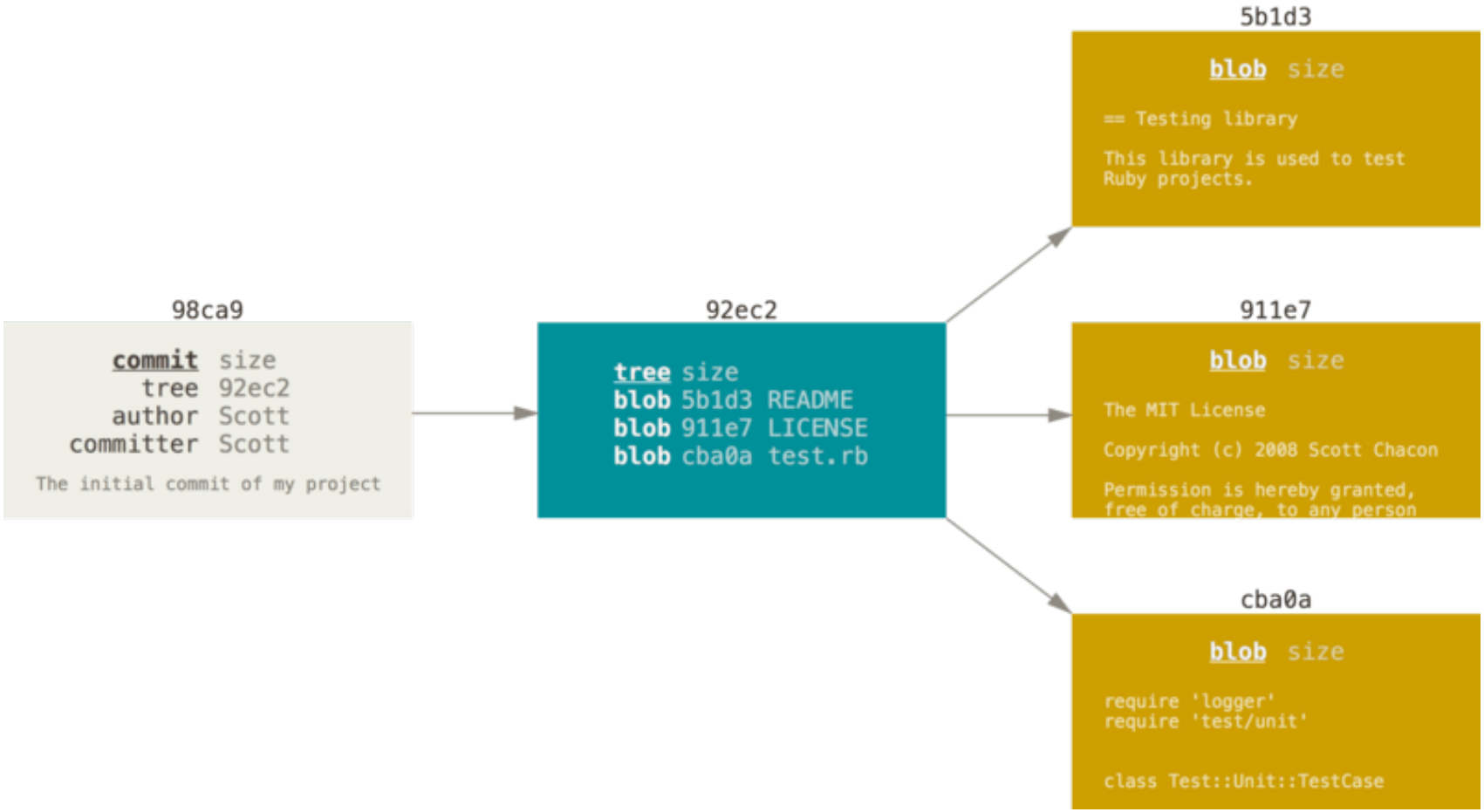

Figure 3: A commit and its tree. *Source*: Chacon & Straub (2021), fig. 9.

What happens when you commit (2)

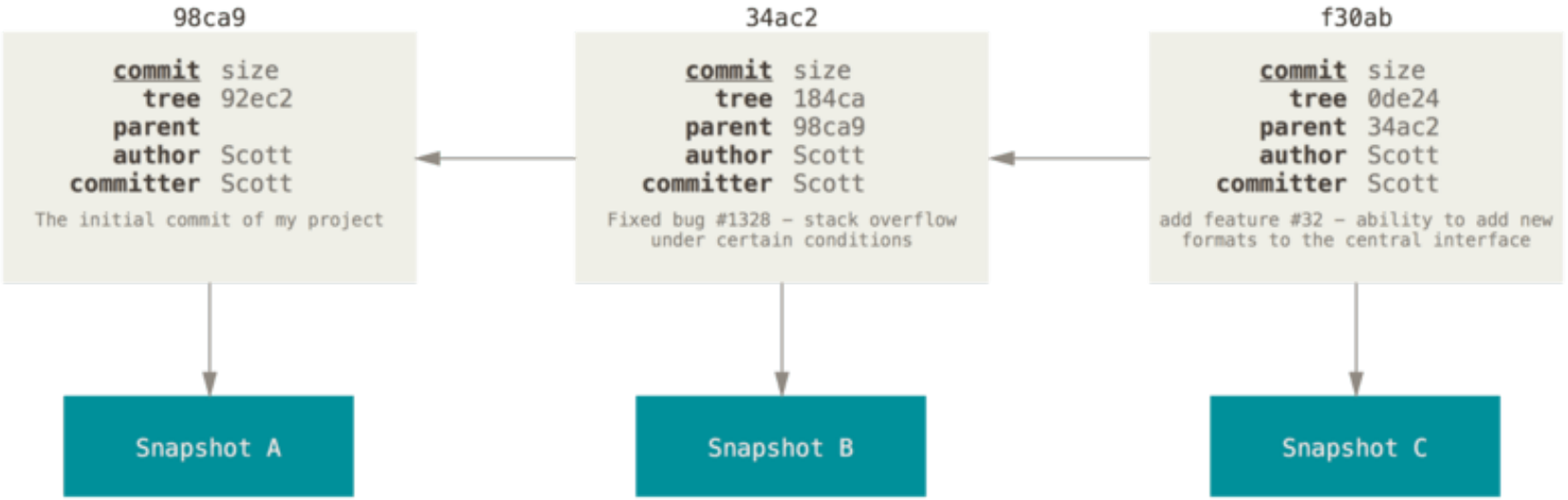

Figure 4: Commits and their parents. *Source*: Chacon & Straub (2021), fig. 10.

Removing and moving files

Removing a file

```
$ git rm PROJECTS.md
```

Moving (renaming) a file

```
$ git mv file_from file_to
```

Note: these operations can also be done manually in your file browser and then staged using `git add`.

Exercise 1

1. Create a new folder.
2. Turn this folder into a Git repository using `git init`.
3. Create a file named `README.md` inside your repo.
4. Check the status of your git repository using `git status`.
5. Start tracking `README.md` using `git add`.
6. Check the status of your git repository again.
7. Modify `README.md` using a text editor (add a line of text for example).
8. Save `README.md`.
9. Check the status of your git repository again.
10. Examine the changes you have introduced using `git diff`.
11. Stage `README.md` using `git add` again.
12. Commit your changes using `git commit -m`

Viewing the commit history

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:  Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:  Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:  Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

Unstaging a staged file

Let's say you would like to unstage `README.md`

```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   modified:  README.md
```

`git status` already tells you what you should do.

```
git restore --staged README.md
```

Unmodifying a modified file

Now `README.md` is unstaged but still modified in your working directory.

```
$git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
      directory)
   modified:  README.md
```

To revert to the last committed version of this file, follow the instructions in `git status`:

```
git restore README.md
```

Danger zone: `git restore` will wipe out all the uncommitted changes.

Exercise 2

1. Go into the repository you created in Exercise 1
2. Examine your commit history using `git log`.
3. Modify `README.md` (add a line of text).
4. Stage `README.md` using `git add`.
5. Check the status of your repo using `git status`.
6. Unstage `README.md` using `git restore --staged`.
7. Check the status of your repo using `git status` again.
8. Unmodify `README.md` using `git restore`.

Working with remotes (1)

When you clone a repository, Git will set up a connection between your **local repository** and the version of this repository on the **server**.

```
$ git clone git@github.com:jolyphil/git-workshop
$ cd git-workshop
```

The repository on the server is called a **remote** repository. By default, the remote repository associated with your local project is called `origin`.

You can see the remotes associated with your project by running the following command:

```
$ git remote
origin
```

Working with remotes (2)

If you check the status of your repo, you will see how your current commit history diverges from your remote.

```
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
```

Important: This is an **offline** command. `git status` does not compare your version history with the version on the server. It compares your local version to the last version of the server **on your machine**.

Fetching from your remote

To update the version history of the server on your machine, you proceed as follows:

```
$ git fetch <remote>
```

`git fetch` does not change in any way your working directory. It simply updates the version history of the server on your computer.

You can compare your local repo with the image of your remote with the following command:

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

Here, `origin` is the remote and `main` is the main branch on the remote.

Pushing to your remote

To synchronize the remote with your local commit history, you proceed as follows:

```
$ git push origin main
```

This is only possible if you have **write access** to the server and **nobody has pushed** since the last time you fetched.

If git rejects your push, do `git fetch`, examine the changes with `git diff`, and, if there are no conflict, `push` again.

We will cover merging problems in the next part of the workshop.

Renaming and removing remotes

To rename remotes, proceed as follows:

```
$ git remote rename pb paul
$ git remote
origin
paul
```

To remove a remote, proceed as follows:

```
$ git remote remove paul
$ git remote
origin
```

Exercise 3

(Through all the next steps: don't forget to use `git status` frequently.)

1. Create a new repo on GitHub with a `README.md` file and clone it on your computer (*demonstration*).
2. Modify `README.md` locally (add a line of text).
3. Stage your changes using `git add`.
4. Commit your changes using `git commit -m`.
5. Try using `git fetch`.
6. Compare your version history using `git diff`.
7. Push your changes using `git push`.