FreeDV Treffen Ende Juni 2022

1. kurzes Willkommen, Status letzte Funkerfahrungen (10 Minuten)

2. **Fragen** und Antworten zu FreeDV (max. 20 Minuten)
3. nächster Termin

4. **FreeDV Entwicklung**

git	Kommando-Zeile
github	web-site
Code-Editoren	

3. Terminplanung

- FreeDV Runde jeden Sonntag
  ab 11:00 Uhr, 40m & 80m
  ab 13:30 Uhr, 40m & 80m
- für unsere Gruppentreffen ist der typische Wochentag **DONNERSTAG**

Juni

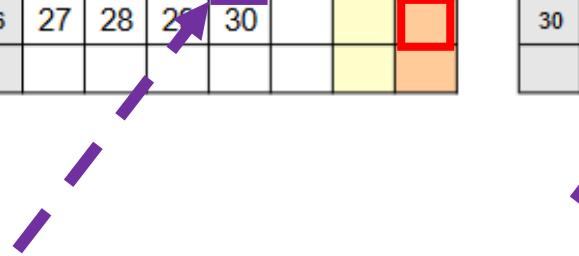
KW	Mo	Di	Mi	Do	Fr	Sa	So
22			1	2	3	4	5
23	6	7	8	9	10	11	12
24	13	14	15	16	17	18	19
25	20	21	22	23	24	25	26
26	27	28	29	30			

Juli

KW	Mo	Di	Mi	Do	Fr	Sa	So
26					1	2	3
27	4	5	6	7	8	9	10
28	11	12	13	14	15	16	17
29	18	19	20	21	22	23	24
30	25	26	27	28	29	30	31

August

KW	Mo	Di	Mi	Do	Fr	Sa	So
31	1	2	3	4	5	6	7
32	8	9	10	11	12	13	14
33	15	16	17	18	19	20	21
34	22	23	24	25	26	27	28
35	29	30	31				

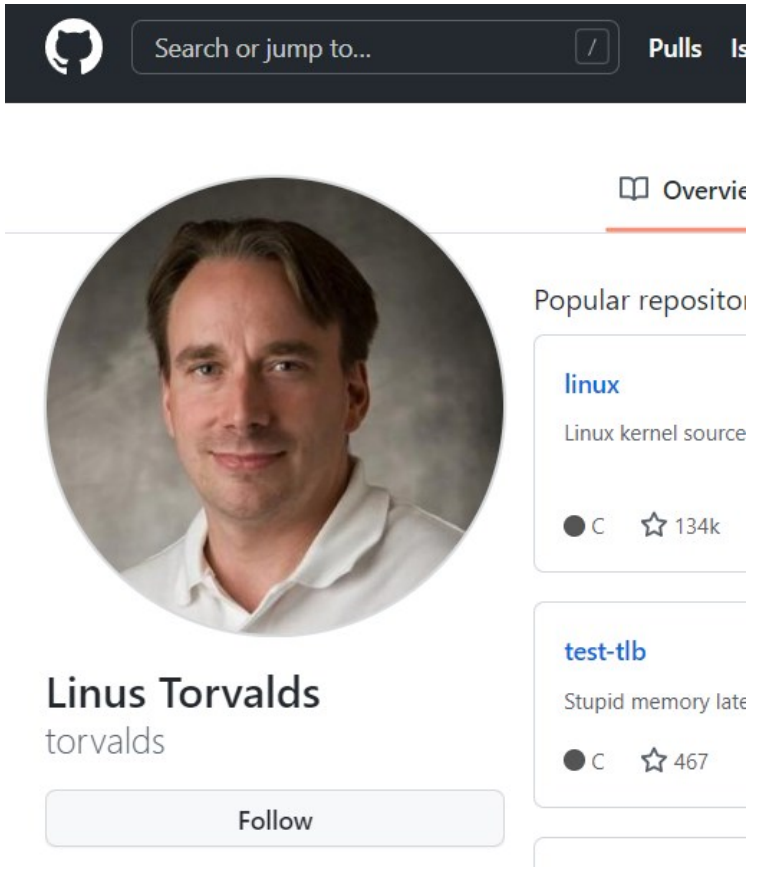


nächstes Treffen in **3 Wochen**

→ **Donnerstag, 14. Juli 2022, 20:00 Uhr**

4. git: lokal und online

- **git** ist primär ein Kommandozeilen-Tool
- natürlich gibt es auch **nette Oberflächen** dazu, aber das betrachten wir heute nicht, wir wollen ja verstehen, wie es funktioniert ☺



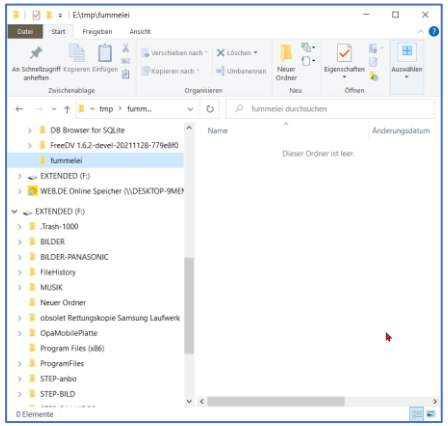
- Installation und Anwendung typisch für Linux („*sudo apt install git*“) aber auch für Windows (https://git-scm.com/download/win)

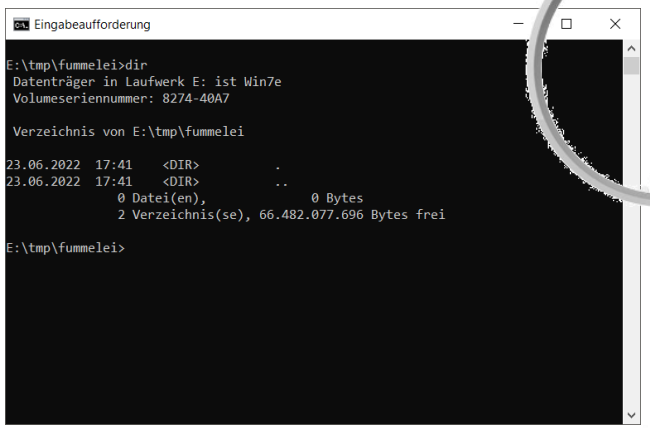
- git arbeitet auf Verzeichnisebene, es gibt immer ein Einstiegs-/Wurzelverzeichnis

4. git: lokal und online

zuerst betrachten wir, wie git „stand-alone“ funktioniert

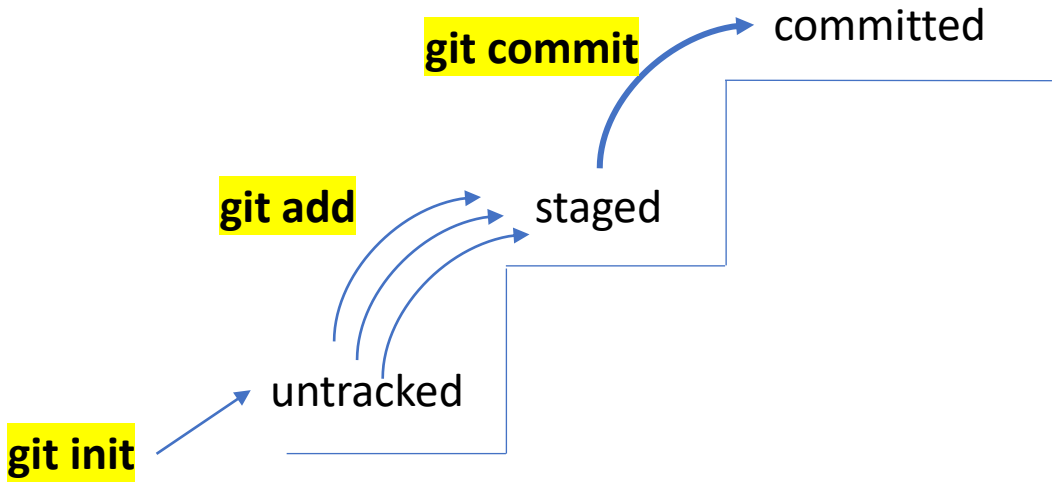
- **git** beobachtet, was ihr in einem Verzeichnis so treibt, nachdem es für das Verzeichnis initialisiert wurde (→ „**git init**“)
- nach einem „git init“ ist git scharf und kann Dateien aus dem Verzeichnis (oder Unterverzeichnissen) aufnehmen und verwalten
- mit einem „**git add**“ teilt ihr git mit, welche Dateien (Verzeichnisse) es aufnehmen soll („git add *“ funktioniert auch…)
- „**git status**“ informiert , was git vom aktuellen Verzeichnis weiß und welchen Status die einzelnen Dateien haben





4. git: lokal und online

- git beobachtet alle Dateien im Verzeichnis und kennt:
  - untracked files
  - staged files (=files to be committed)
  - committed files



- „**git add**“ übernimmt den aktuellen Dateiinhalt in einen „Vor-Freigabe-Bereich“ (staging area)
- „git add“ kann mehrmals ausgeführt werden, um weitere Dateien in die staging area zu übernehmen
- ein erneutes „git add“ mit einer bereits hinzugefügten Datei aktualisiert diese in der staging area

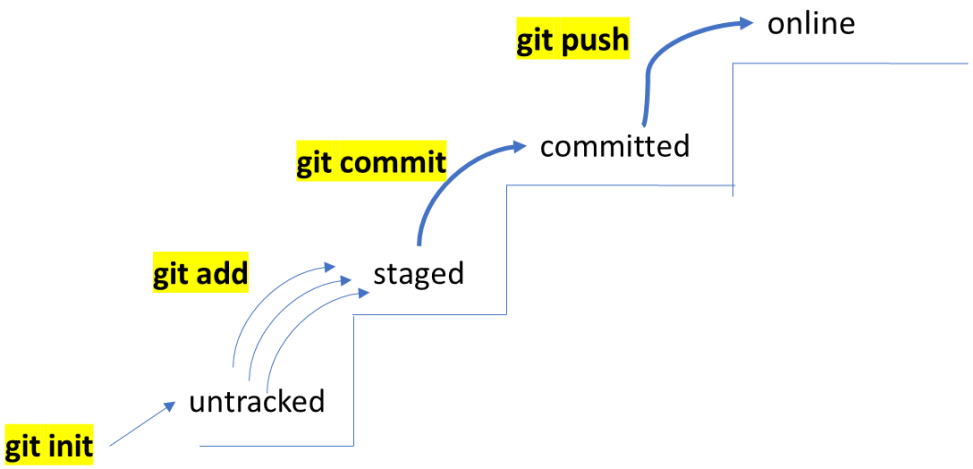
- „**git commit –m *„Kommentar“*** “ nimmt alle Dateien gemeinsam aus der staging area und fügt sie zu den freigegebenen Dateien hinzu, nur sinnvoll mit einen Kommentar

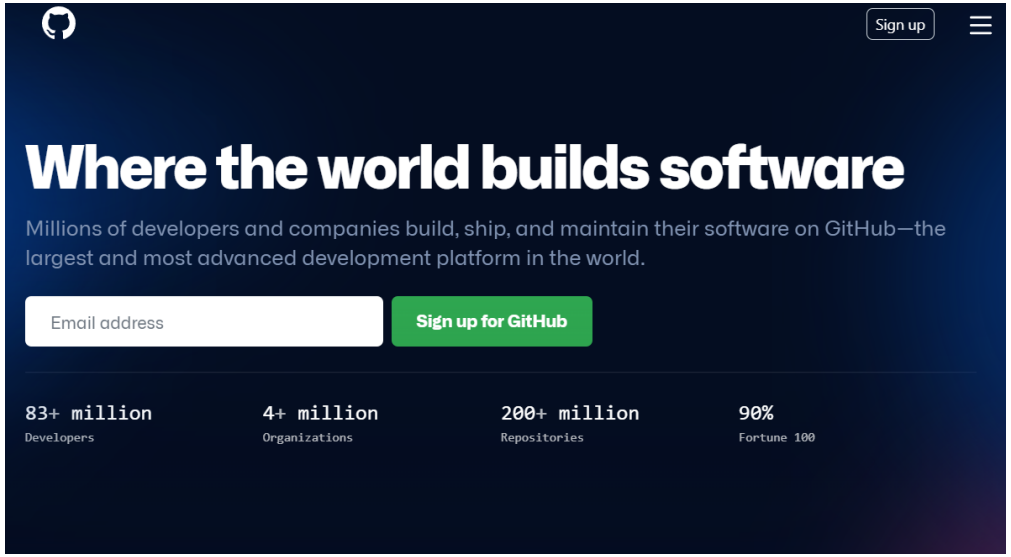
- „**git log**“ sagt etwas über den aktuellen Verzeichnis-Status,
  „**git log –p**“ liefert mehr Details über die Änderungen an einzelnen Dateien,
  „**git status**“ ist immer auch eine gute Idee

4. git: lokal und online

- anders als Subversion arbeitet **git** auch rein lokal, das haben wir bisher gemacht
- wenn wir mit anderen zusammen arbeiten wollen, dann brauchen wir einen Online Account auf einem geeigneten Server (verschiedene Möglichkeiten)
- wir versuchen das heute direkt mit **https://github.com**
- erster Schritt: einen Account anlegen → sendet eine Bestätigungs-Email

- nächster Schritt „**git push** https://github.com/LoginName/RepoName“
- repo: kurz für Repository

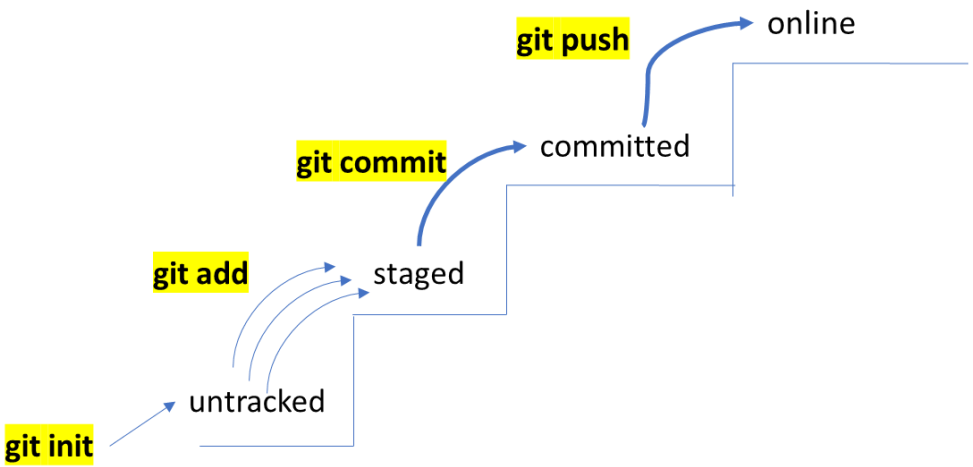


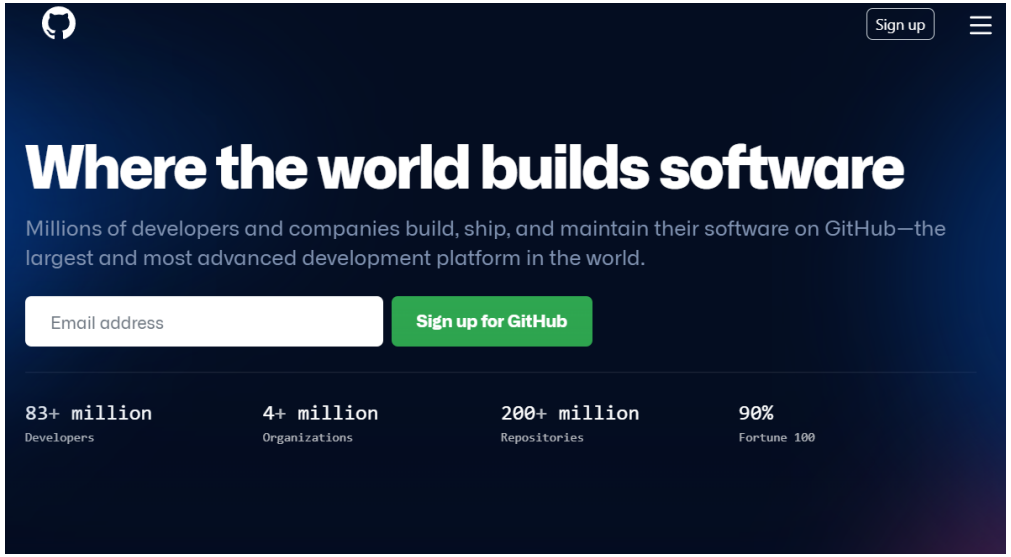


4. git: lokal und online

- anders als Subversion arbeitet **git** auch rein lokal, das haben wir bisher gemacht
- wenn wir mit anderen zusammen arbeiten wollen, dann brauchen wir einen Online Account auf einem geeigneten Server (verschiedene Möglichkeiten)
- wir versuchen das heute direkt mit **https://github.com**
- erster Schritt: einen Account anlegen → sendet eine Bestätigungs-Email

- nächster Schritt „**git push** https://github.com/SittingBull18/a“
- repo: kurz für Repository

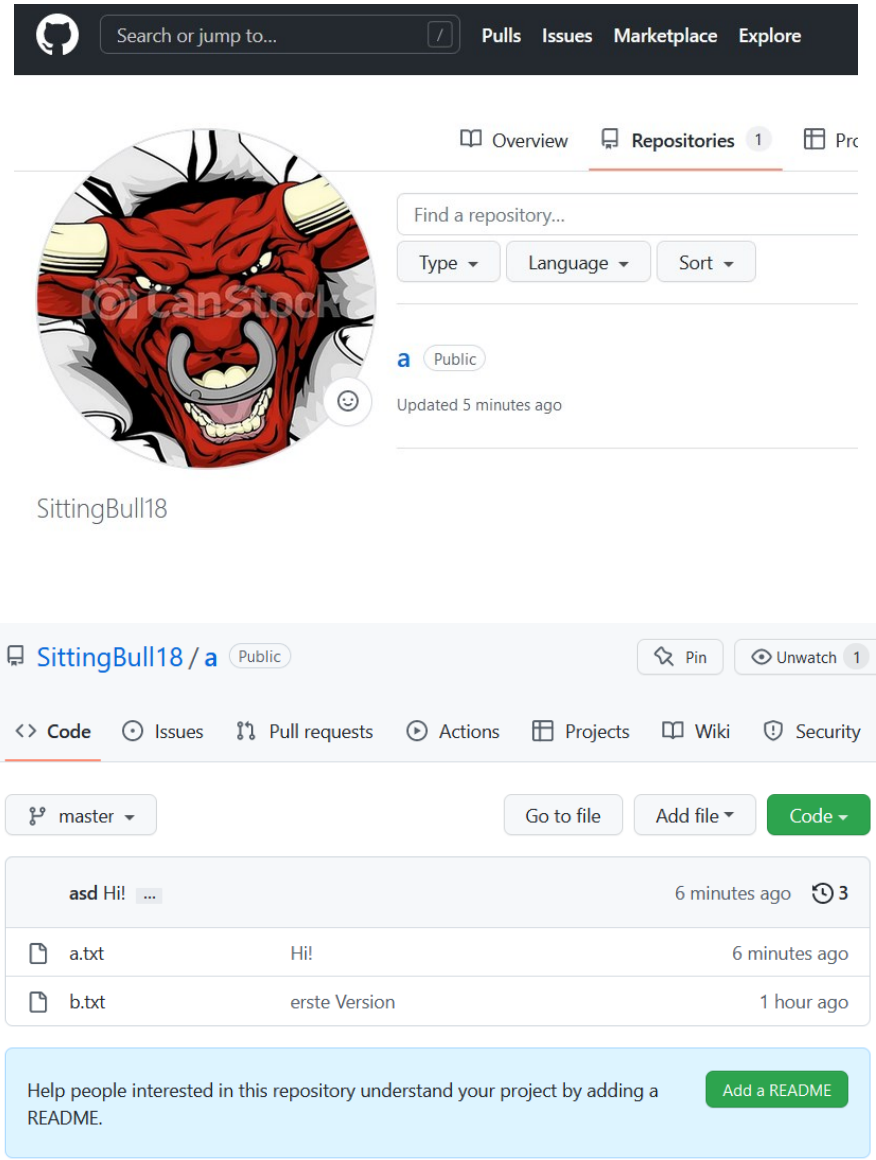




4. git: lokal und **online**

- „**git push**“ erzeugt einen Datentransfer zum angegebenen Server (hier github)

- „**git clone**“ ist das Pendant dazu und transferiert das angegebene Repo auf den lokalen Rechner (legt ein neues Verzeichnis mit dem entsprechenden Namen an)
- das ge-clonte Verzeichnis funktioniert wie zuvor („git status“, …)
- wer Schreibrechte für das Repo hat, kann direkt veränderte Dateien ins Repo push-en
- das probieren wir direkt mal aus

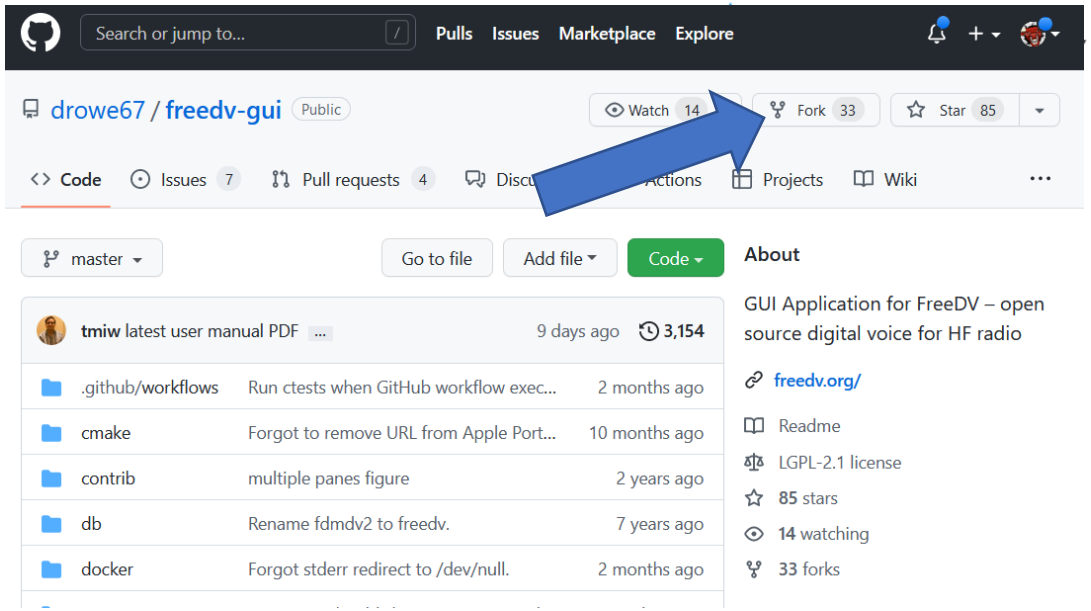


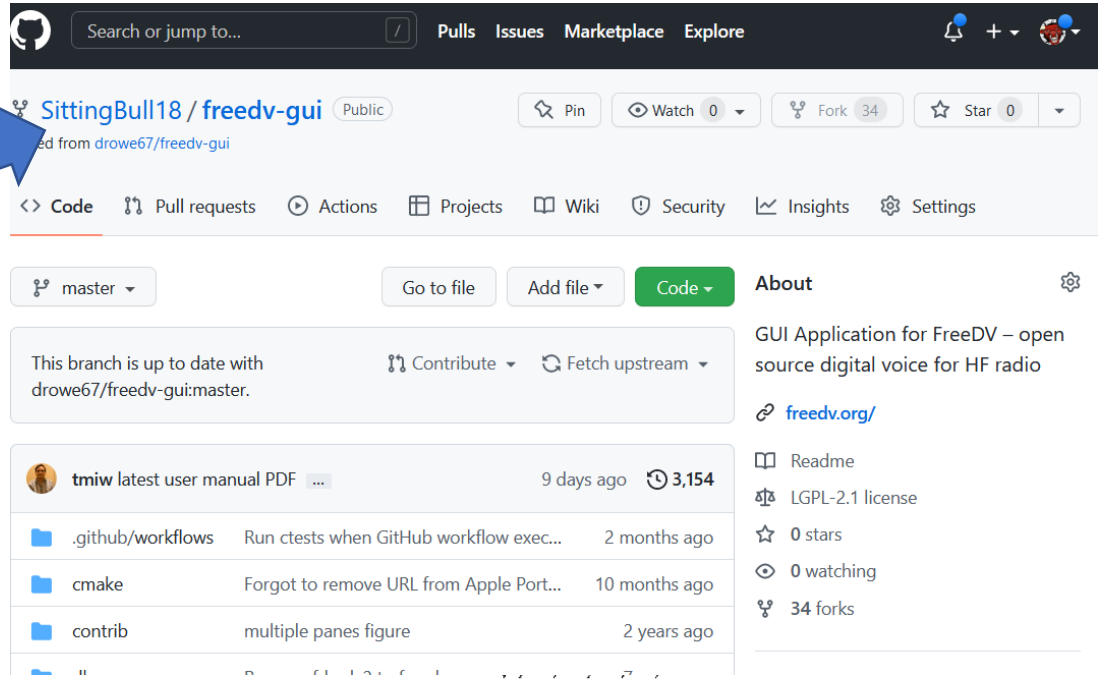
4. git: lokal und **online**

- „**git clone**“ funktioniert mit allen öffentlichen Repos
- „**git push**“ funktioniert nur bei Repos mit eingeräumten Schreibrechten, und das macht ja auch Sinn, sonst geht ja alles drunter und drüber!

- wie arbeitet man dann zusammen?

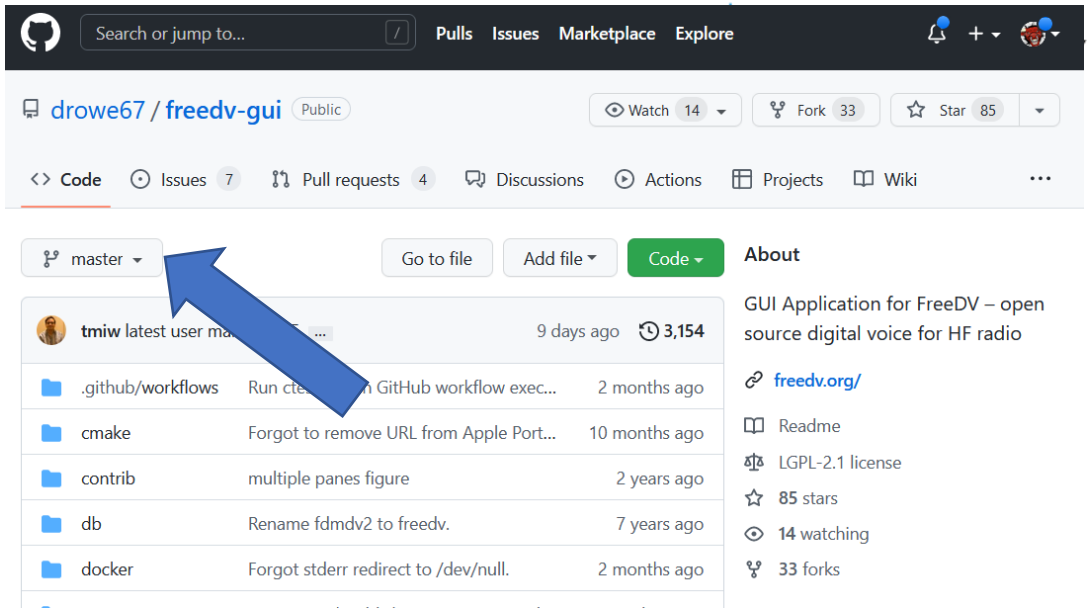
- indem man ein öffentliches Repo „**forked**“
- auf „meinen fork“ habe ich beliebigen Zugriff, kann mir „meinen fork“ clonen, editieren und pushen
- ge-push-te Änderungen gebe ich dann mit einem **Pull-Request** zurück an den Verwalter des Quell-Repos zur Durchsicht und Freigabe





4. git: lokal und **online**

- „**branch**“ gibt es auch noch
- das ist hilfreich für verschiedene, parallel laufende Entwicklungen in einem Repo, die sich nicht direkt beeinflussen sollen



- „**git branch - - list**“ zeigt die verfügbaren Branches des gewählten Repos an
- oder auch sichtbar auf github

- „**git checkout branch**“ wählt einen bestimmten Branch aus, danach braucht es wieder ein „git clone“ um die entsprechenden Dateistände herunter zu laden

4. git: lokal und online