Git - Software-Architektur

5-10 Min Präsi

→ Softwarearchitektur analysieren
→ Stile und Muster identifizieren → dynamische Sicht
→ Sinnvolle Sichten und Diagramme
→ Vorteile Nachteile → Bewertung → Statistik

Allgemeines zu Git:

- freies Open-Source verteiltes Version-Control-System
- für kleine bis große Projekte geeignet
- schnell und effizient

C	48,5%
Shell	35,9%
Perl	6,8%
Tcl	4,6%
Python	2,0%
Makefile	0,8%

Git-Interna:

I Git-Objekte

- Git ist ein inhaltsadressierbares Dateisystem.
- der Kern von Git ist ein einfacher Key-Value-Speicher
- jede Inhalt kann in ein Git-Repo eingefügt werden → Git gibt einen eindeutigen Schlüssel zurück, mit dem der Inhalt abgerufen werden kann
- a content-addressable filesystem with a VCS user interface written on top of it
- Objekttyp blob: file, hat SHA-1, ohne Speicherung des Dateinamens, nur dessen Inhalt
- Objekttyp tree: Ähnlich zu Unix-Filesystem. Ist ein Ordner. Hat SHA-1 hash
- ~~Commit~~ Objekttyp Commit: Führt mehrere trees und blobs in einem zeitlich zusammen. Sonst müsste man sich alle SHA-1-Namen merken.

II Git References

- statt sich eine Commit ID zu merken, speichert man sich bestimmte Punkte als Referenzen, d.i. Pointer mit einem einfachen Namen
- Remotes:
- HEAD: Zeigt auf aktuellem Branch, mal auch Git Obj.
- Tags: auch ein Git Object, points to commit

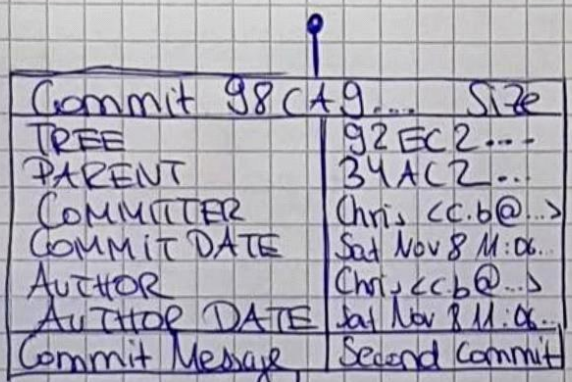
Git-Verzeichnis

- .git/
  - hooks/: Beinhaltet Client- oder Server-seitige Hook-Skripte
  - info/: exclude-Datei, ähnlich zu .gitignore
  - description: Zu vernachlässigen
  - config: Projektspezifische Konfigurationen

- HEAD: Points auf ausgecheckten Branch
- index: Hier speichert Git Informationen der Staging Area
- objects/: Speichert alle Inhalt der Datenbank
- refs/: Pointer zu Commit-Objekten werden hier gespeichert (Branches, Tags, Remotes...)

Commit-Aufbau

- ~~Sequenzdiagramm~~



Porcelain & Plumbing

- Plumbing-Kommandos sind Git-interne Kommandos, mit denen man nie etwas macht ~~git commit-tree, git cat-file, git-hash-object, git write-tree, git update-index...~~
- Porcelain-Kommandos sind alle anderen Kommandos, git pull, ...

Hat Git Design Patterns?

Motivation

⇒ keine wirklichen

⇒ Linus Torvald hat Git geschrieben. Linus hat hier nie auf Design Patterns geachtet (siehe auch Linux ⇒ das ist vor dem Buch entstanden!)

⇒ A programmer like Linus doesn't intentionally say "I'm going to use a decorator pattern". Es kann passieren, dass zufällig etwas ähnliches entsteht, aber nicht geplant.

⇒ Git ist zum großen Teil in C geschrieben

⇒ Linus hat C++ rejected, aus dem Grund würde er wohl nie Code à la Python schreiben. Diese Idee könnte ihn sogar anwidern

⇒ Linus hat aber eine starke Opinion über System sicht die Dinge und Git wurde auch so geschrieben

---

Architektur

Auffälligkeiten →

⇒ Git ist ähnlich einem OS-Filesystem.

⇒ Client-server

⇒ Git wird über feste Kommandos über das Command Line Interface bedient

⇒ Architektur-muster:

↳ hier gibt es Plumbery und Porcelain Kommandos.

Peer-to-Peer Pattern

↳ Das Game ähnelt etwas dem Façade, da der User die CLI meist mit Porcelain-Kommandos bedient, man könnte auch mit vorhandenen Plumbery Commands verwenden, deren Client sind jedoch tief in Git drin.

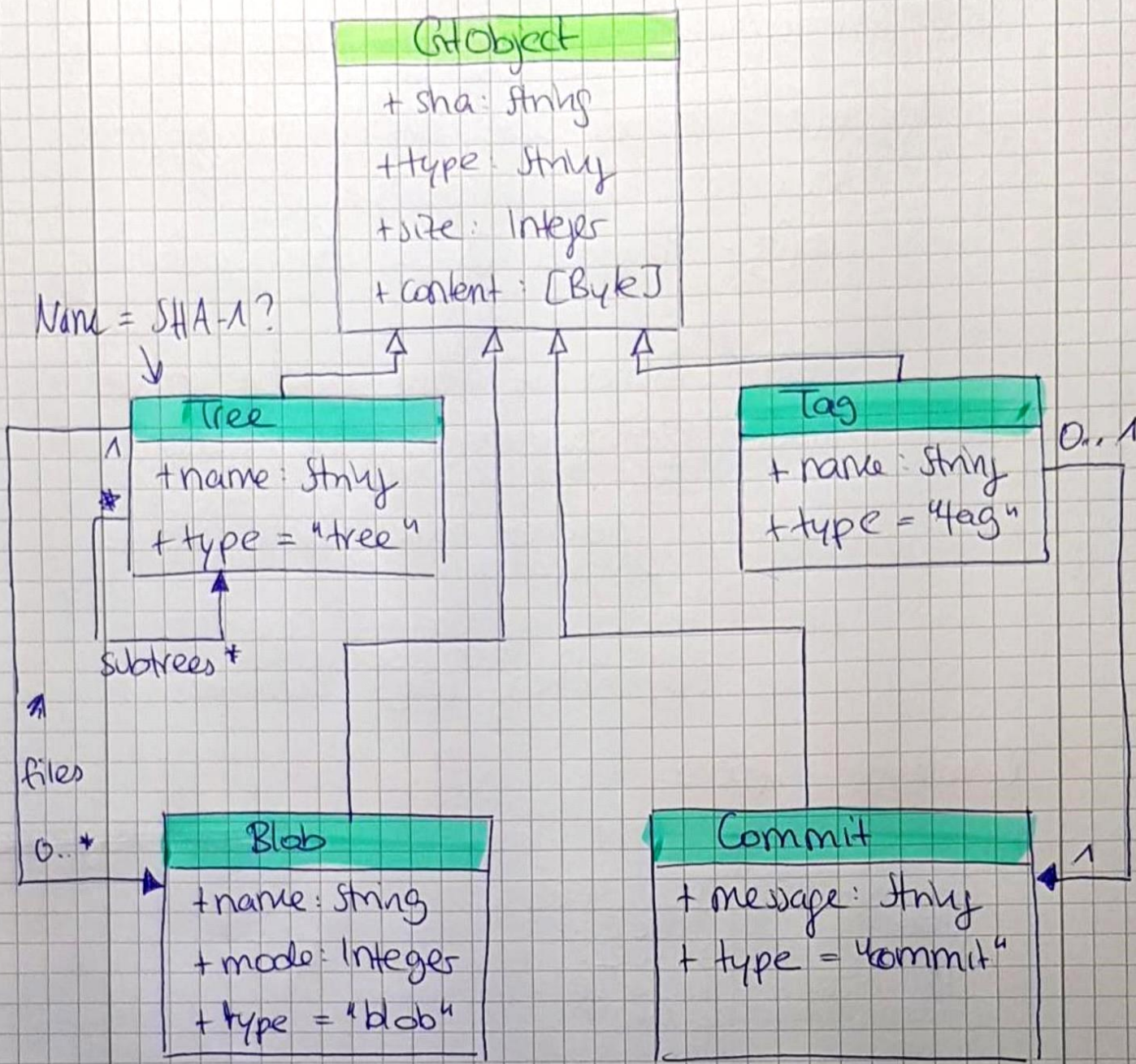
Composite? Command? Workflow

mit lokal Repo

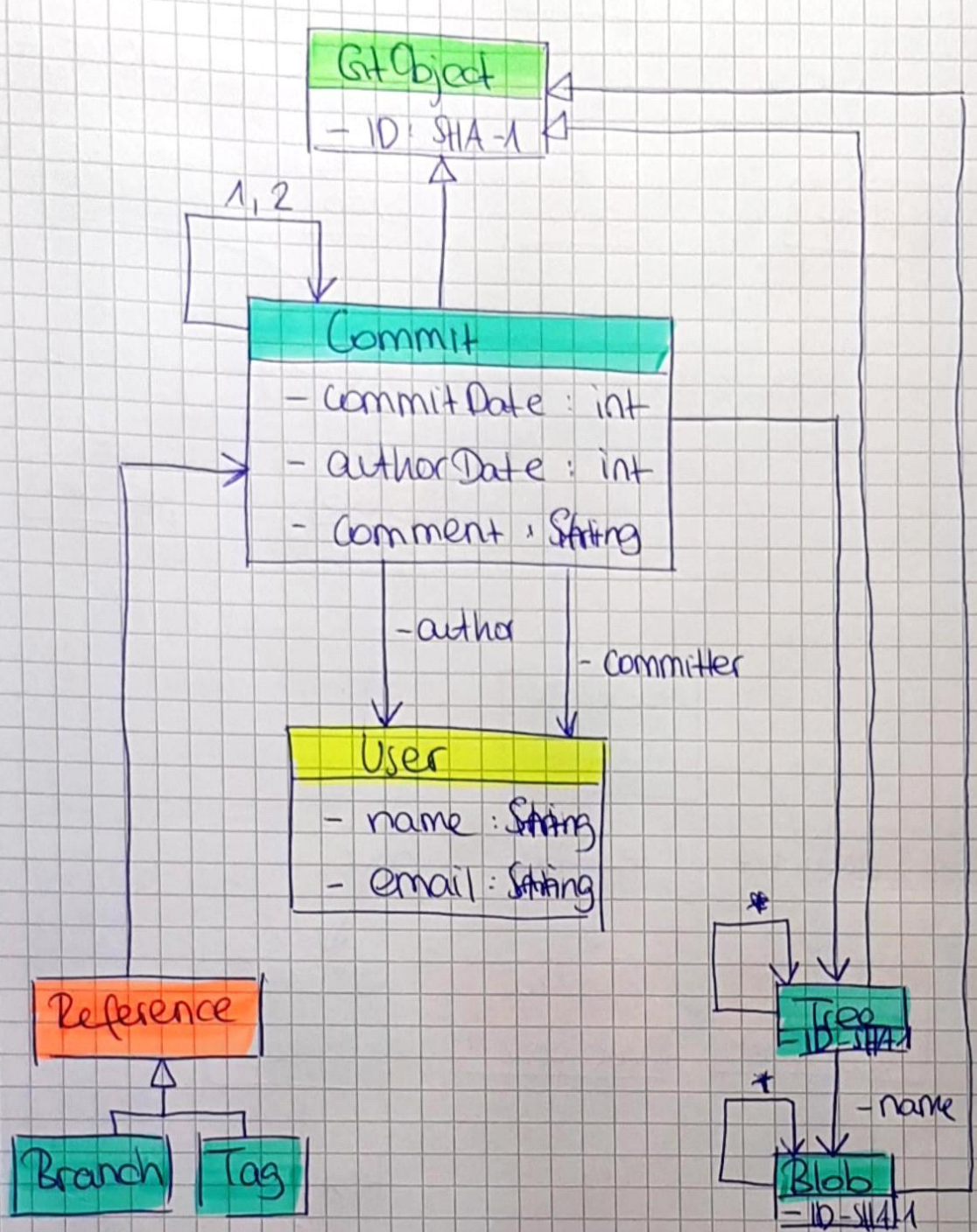
⇒ Git ähnelt einem Blockchain bzw einer linked List

⇒ Dezentralisiert, Transactions immutable, Transactions independently verifiable

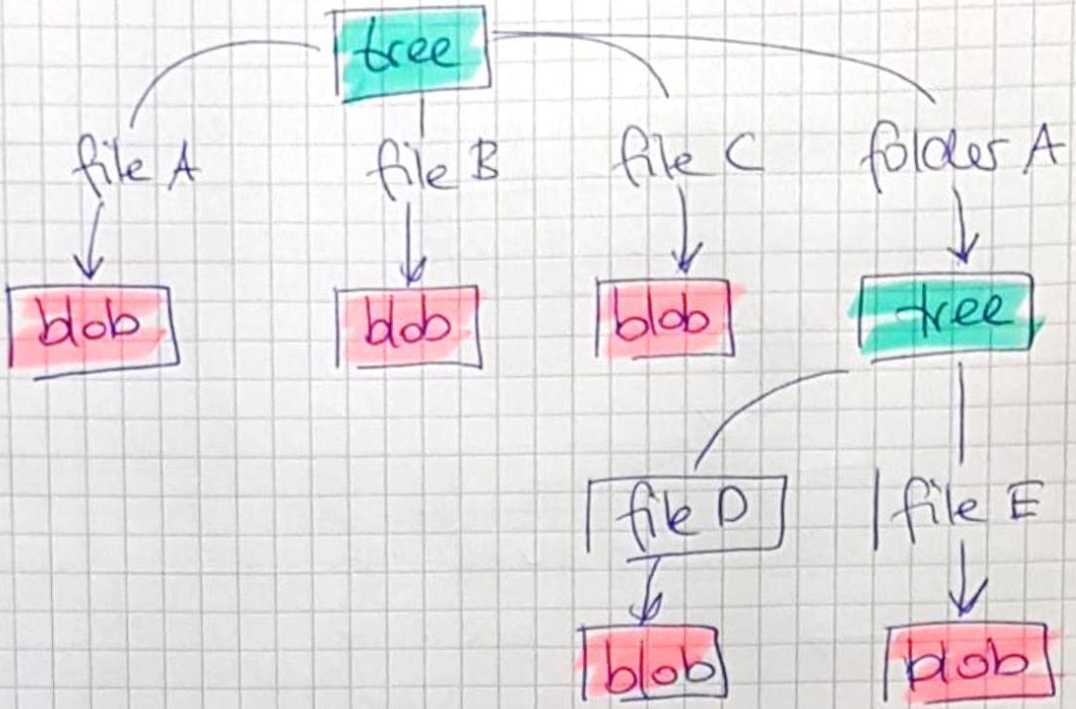
The Object Database

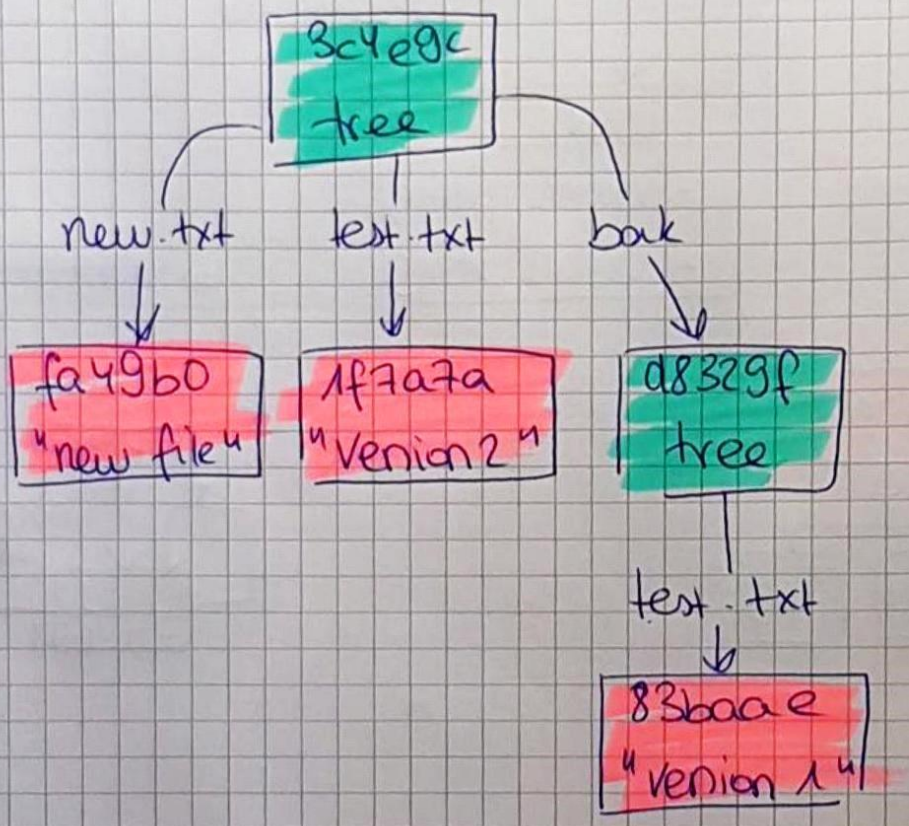


Git Data Model

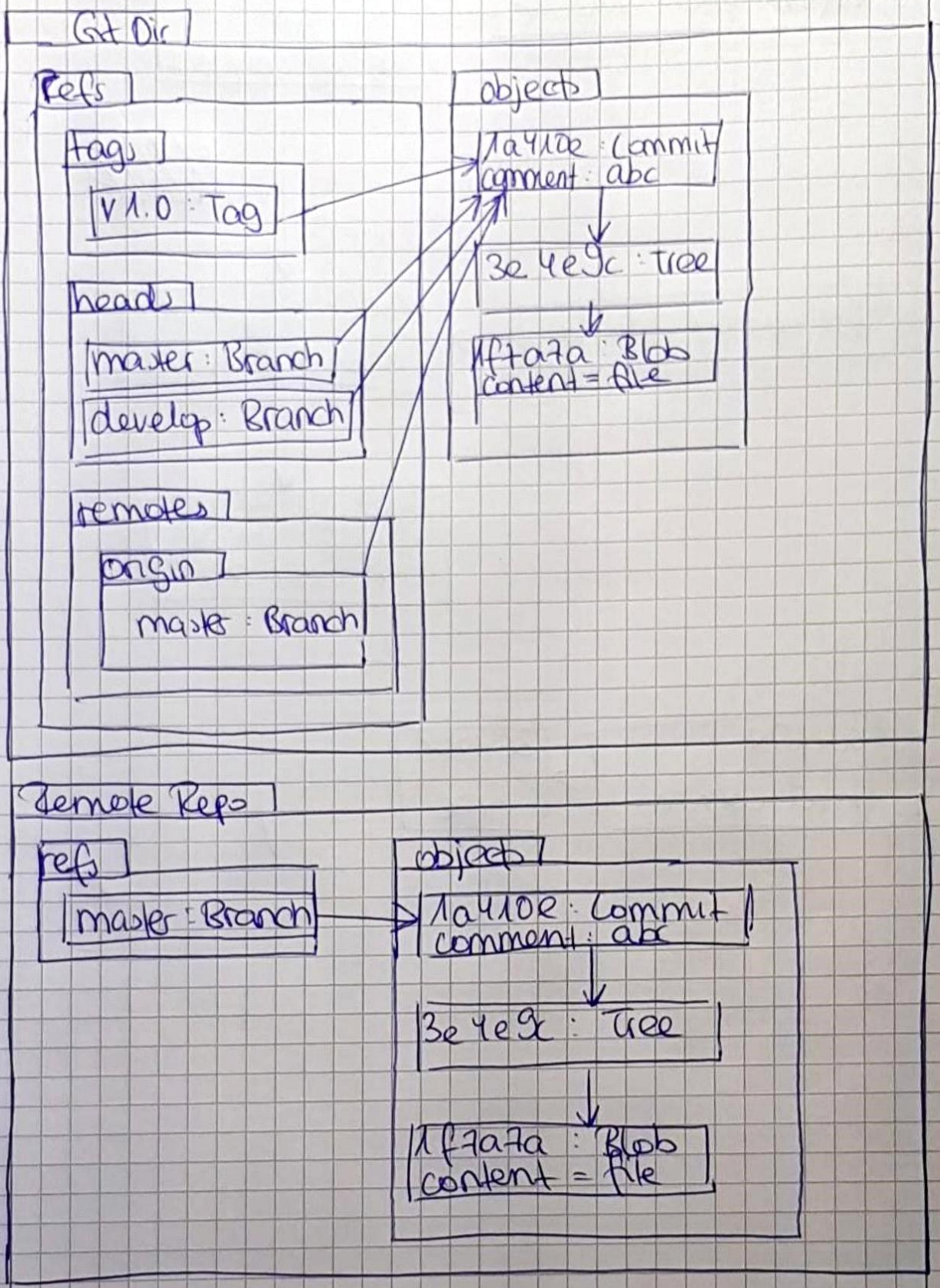


Git Objects


tree
file A
file B
file C
folder A
blob
blob
blob
tree
file D
file E
blob
blob


3c4e9c
tree
new.txt
test.txt
bak
fa49b0
"new file"
1f7a7a
"version 2"
d8329f
tree
test.txt
83baae
"version 1"

Git Objects (Git References)



~~The Object Database~~ Git References