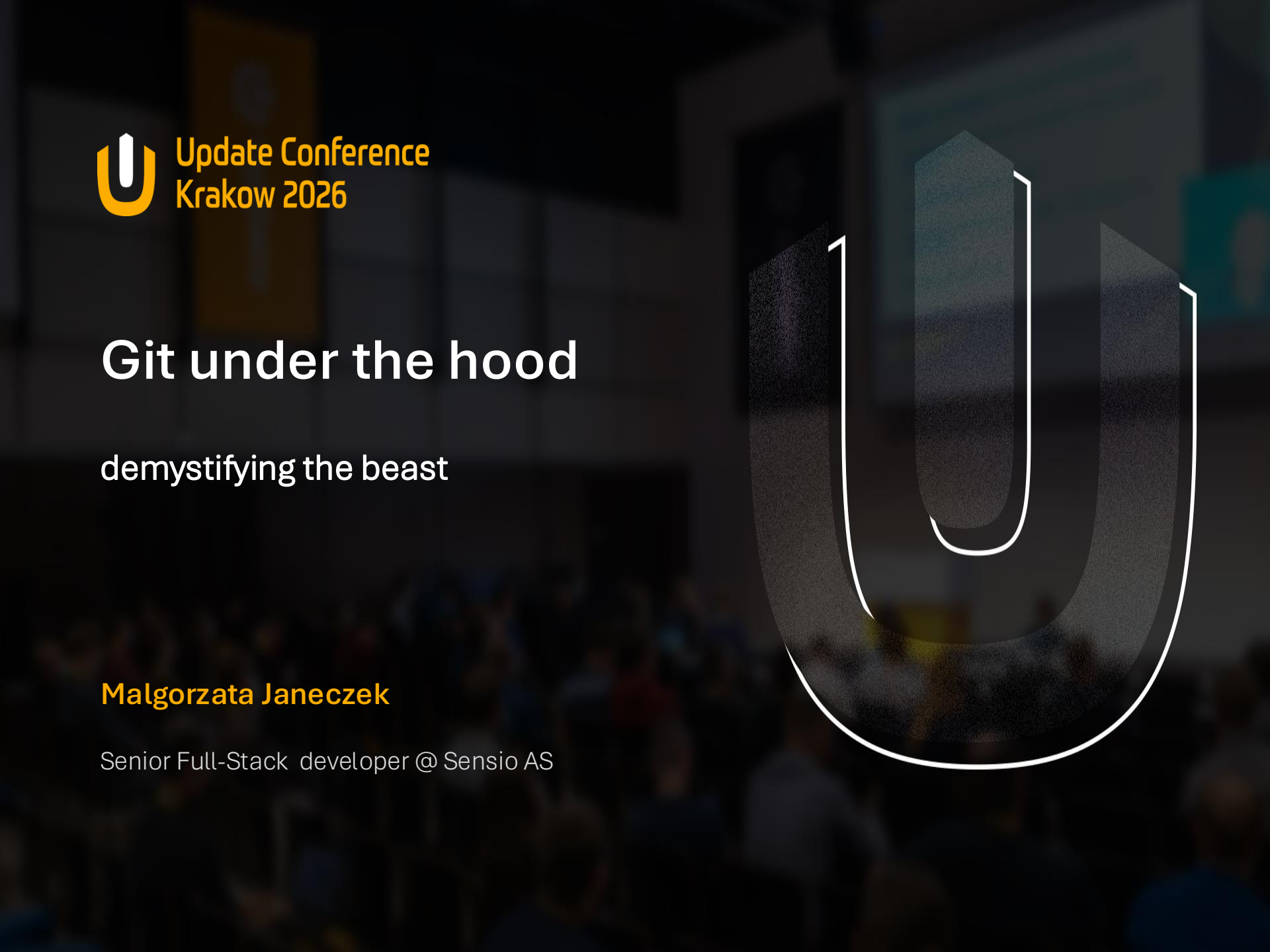


Git under the hood

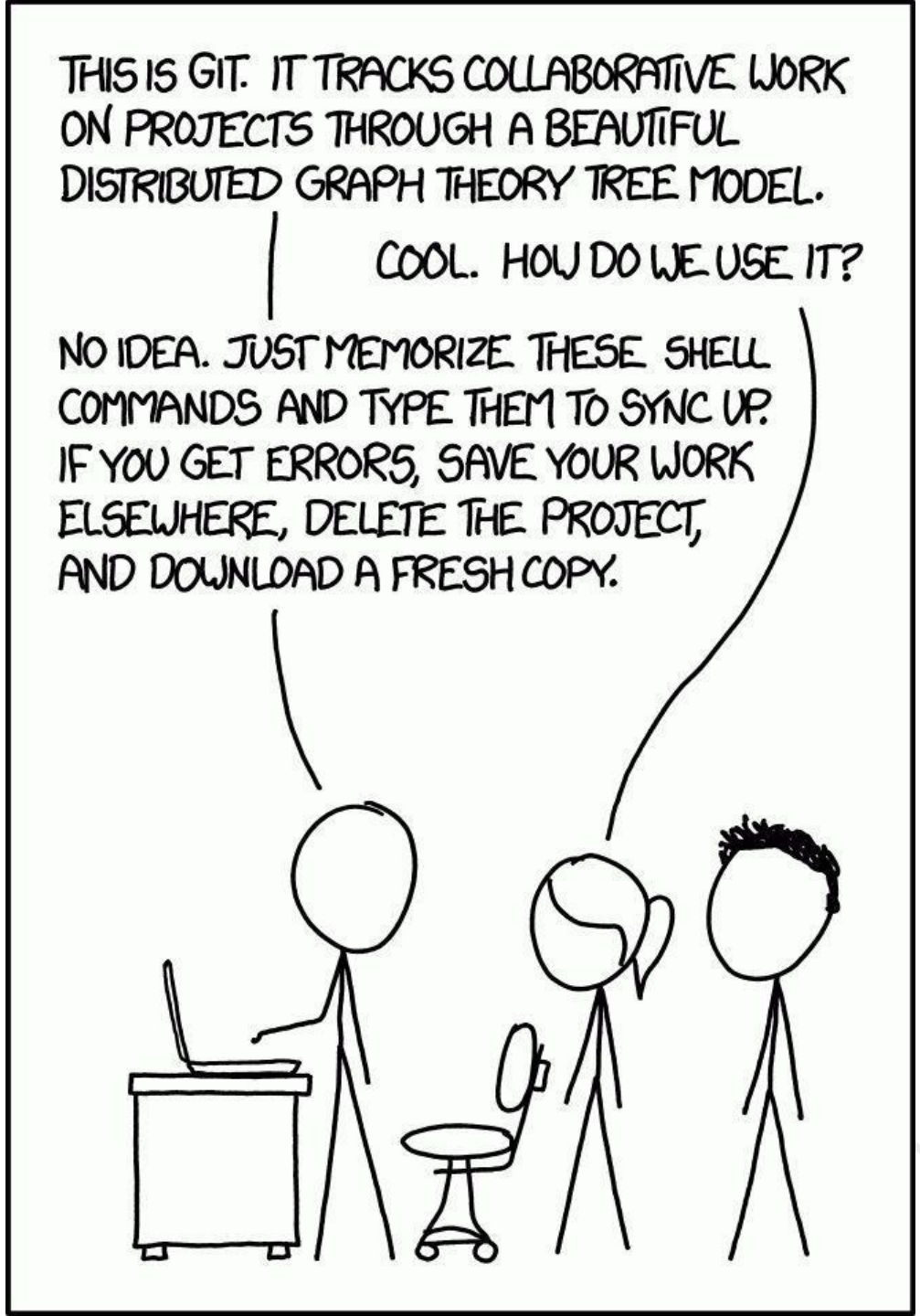
demystifying the beast

Malgorzata Janeczek

Senior Full-Stack developer @ Sensio AS

Let me start with a question...

The panic move


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git is “**not just a version control system**”.
It’s a very small database with two features:

- it stores content
- and it stores relationships between that content

Hi, I’m Gosia

Sr. Full-Stack .NET / React Developer @ Sensio

Co-author of DevOps-journey blog devopsifyme.com

DEVOPSIFY ME

sensio

ITM

JetBrains Community Contributor

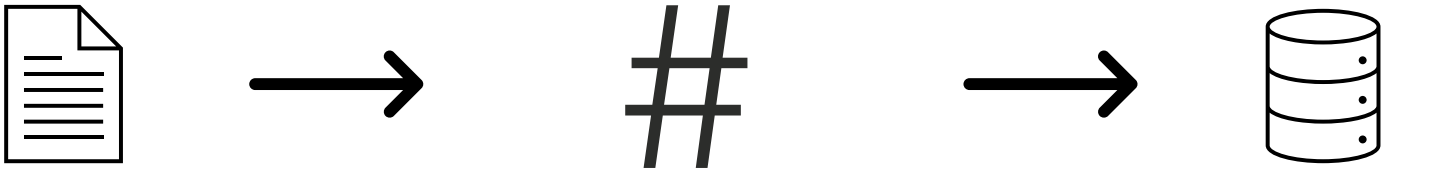
Małgorzata < JANECZEK />


git
deploy?
later

🎭 Act 1

The «memory machine»

🧠 Core idea

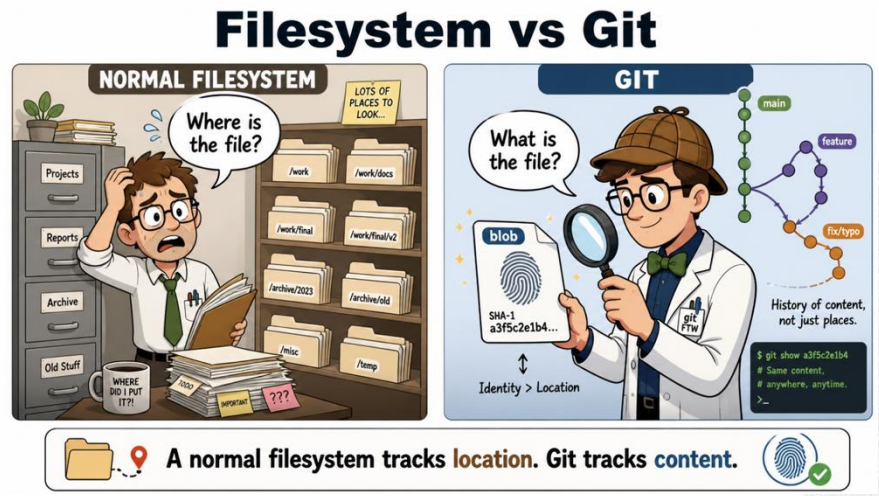


Summary

1. Same content = same hash
2. Where the object lives

```
Hash:
ce013625030ba8dba906f756967f9e9ca394464a

Stored as:
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a
```



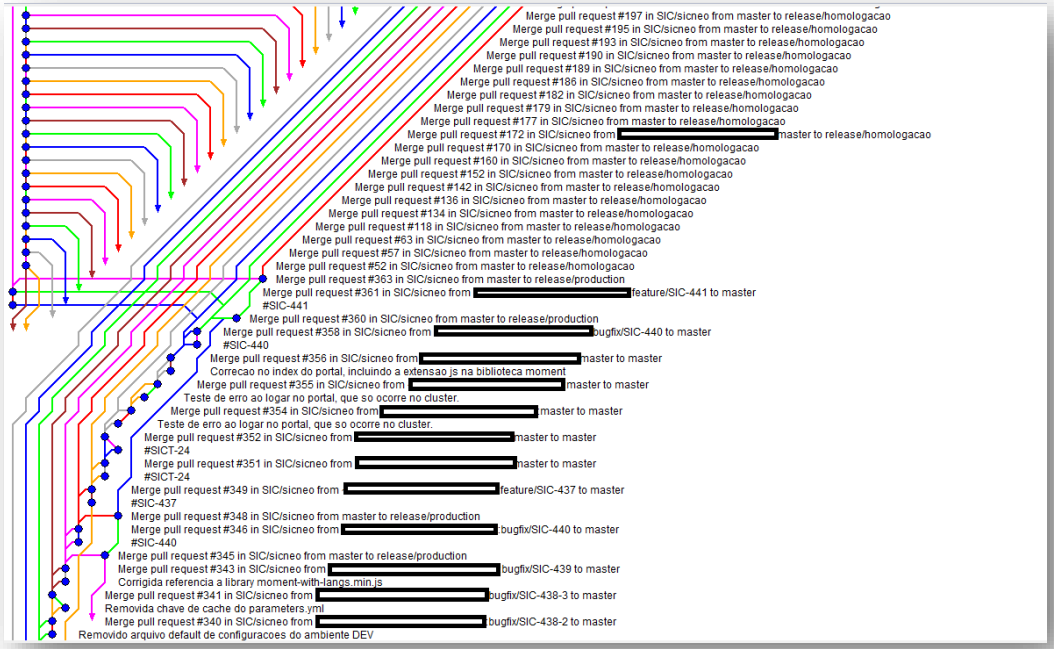
End of Act 1

Act 2

Git learns folders (Trees)

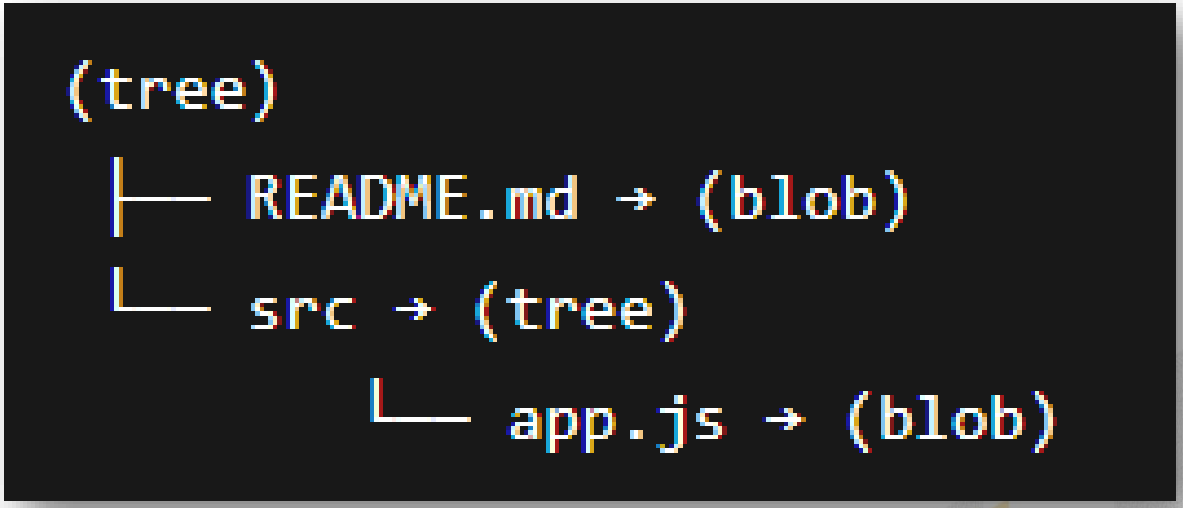
Core idea

- A tree is how Git represents a directory.
  - Tree = folder
  - Blob = file
  - Tree can point to blobs and other trees



Source: some old Reddit thread

But it can be simplified



Summary

- `100644 blob 47d6a0ec8bf034eb64caa2c7c36599852172e3ac README.md`
  - Blob -> file content
  - <hash> - where content lives
  - README.md - File name
- `040000 tree dfecde582fbd595755b60500b8389487c7e29833 src`
  - Tree -> folder
  - <hash> - another tree
  - Src – folder name

End of Act 2

Act 3

Git learns history


dev
bugfix/login
git
commit
coffee
repeat
Pro Git
a1b2c3d
initial commit
2023-01-01
The beginning of everything ♡
d4e5f6g
add login page
2023-01-03
First real step!
b7c8d9e
finally works
2023-01-15
Worth it. ♡
e9f0a1b
ship v1
2023-01-20
Here we go. ♡
f1a2b3c
fix nasty bug
2023-01-07
That bug haunted me
c0d1e2f
refactor auth flow
2023-01-10
Cleaner. Stronger. ♡
1a2b3c4
improve tests
2023-01-18
Future me thanks me.
5d6e7f8
small docs update
2023-01-22
Little things matter.
feat
auth
refactor
tests
docs
If trees are photos,
commits are a photo album. ♡

🧠 Core idea

- tree → what the project looks like
- parent → what came before
- metadata → who/when/message

- 👉 A commit is a snapshot… with memory.

💡Summary

- Commits store snapshots NOT diffs

📁 Blob = content
🌳 Tree = structure
📸 Commit = snapshot in time

End of Act 3

🎭 Act 3 (bonus)

Git learns efficiency (Packfiles)

🧠 Core idea

Git stores objects efficiently using packfiles.

Loose objects
→ compressed
→ packed together
→ delta-compressed internally

💡 Summary

Git snapshots look expensive...
but internally Git aggressively compresses objects.

That is why:

- repositories stay surprisingly small
- cloning is fast
- history is cheap

🎭 Act 4

Branches (just pointers)

🧠 Core idea

- A branch is just a pointer to a commit.
- Creating a branch does NOT copy anything.

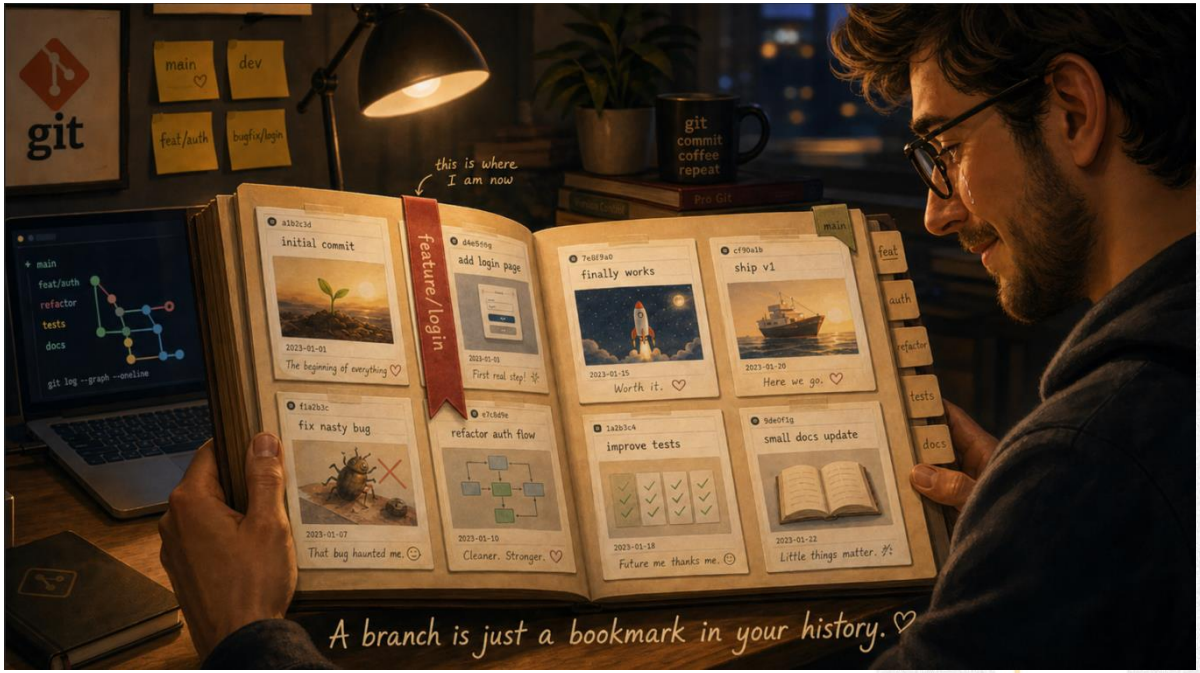
main: A -> B -> C

feature: -> D

🎯 Branches are cheap because they are just pointers.

Summary

- Branch = pointer
- Branch != copy
- Branches lives in .git/refs/heads



End of Act 4

🎭 Act 5

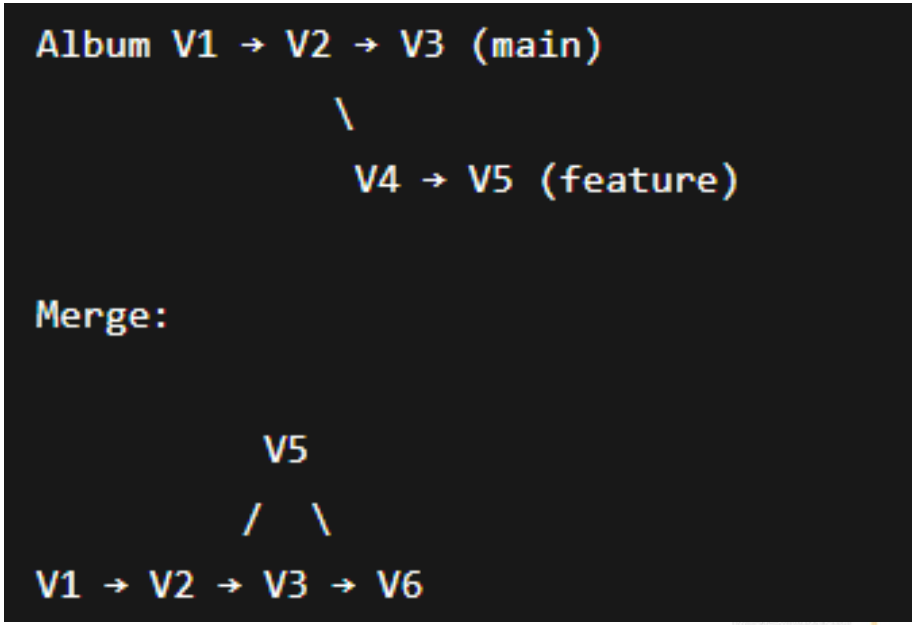
Merging (combining timelines)

main: A -> B -> C

feature: -> D

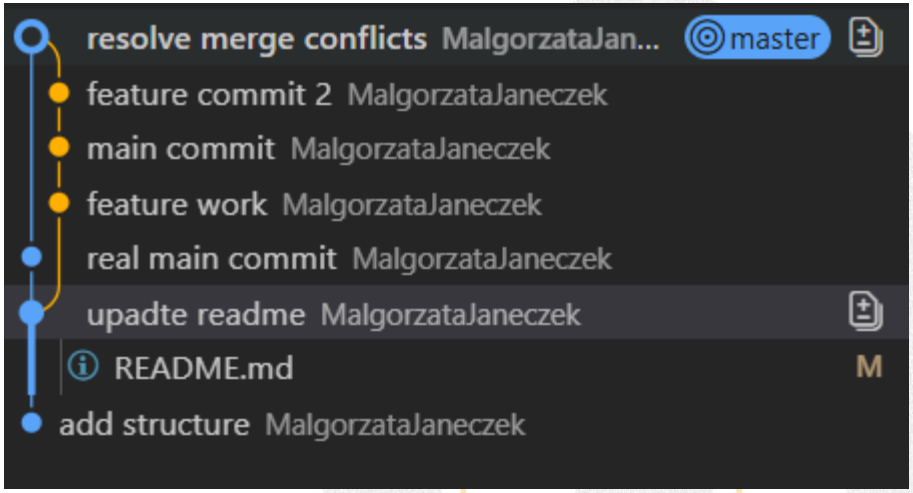
🧠 Core idea

- A merge is just a commit with TWO parents.
  - normal commit → 1 parent
  - merge commit → 2 parents



Summary

- Merge = new commit
  with two parents
- Merge connects histories
  It does NOT rewrite them
- If conflict – you must resolve it!



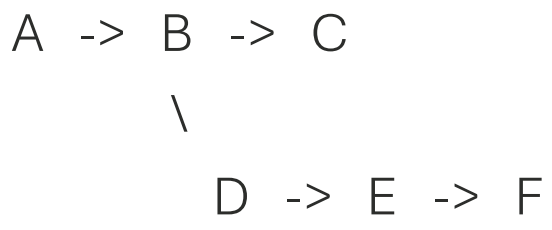

dev
bugfix/login
git
commit
coffee
repeat
Pro Git
this is where
I am now
main
line of work A
a1b2c3d
initial commit
2023-01-01
The beginning of everything.
c3d4e5f
add login page
2023-01-02
First real step!
f6a7b8c
add auth flow
2023-01-03
Stronger foundation.
continuing together
n4e5x6t
ship v1
2023-01-11
Here we go.
two lines of work
m3r9e0
merge feature/login
2023-01-10
Stronger together.
one shared history
p7u8r9e
tests & polish
2023-01-12
Quality first.
feature/login
line of work B
d1e2f3g
fix nasty bug
2023-01-07
That bug haunted me.
e4f5a6b
improve tests
2023-01-08
Future me thanks me.
f7g8h9i
small docs update
2023-01-09
Little things matter.
q1w2e3r
docs & release notes
2023-01-13
For future us.
A merge brings two histories together.

End of Act 5

Act 6

Rebase recreates history

What if your history gets messy



🧠 Core idea

- Rebase takes your commits...
  and replays them on top of another branch.

- Rebase does **NOT** move commits.
  It creates new ones

Summary

- merge → connect timelines
- rebase → rewrite timelines
- It’s a tradeoff:
  - + clean history
  - - rewritten history

REBASE: moving your work to a new starting point

End of Act 6

🎭 Act 7

Merge + Squash (clean history without rewriting)

🧠 Core idea

- Squash merge takes all the work from a branch… and turns it into ONE commit.

- A squash merge is NOT a real merge commit.
  - normal merge → 2 parents
  - squash merge → 1 parent

💡 Summary

- Keep the result, drop the internal history
- Squash ≠ merge commit
  Only ONE parent

SQUASH: many commits, one memory

BEFORE
Your feature branch has many commits...

main


a1b2c3d
Initial commit


d4e5f6g
Add login


h7i8j9k
Improve UI


z9y8x7w
Refactor auth

feature


l1m2n3o
Add export


p4q5r6s
Fix edge case


t7u8v9w
Add tests


x1y2z3a
Update docs

Lots of little steps to get the job done.

SQUASH
Git combines them into a single commit...

SQUASH turns many commits into one clean memory.

AFTER
A single commit keeps history clean.

main


a1b2c3d
Initial commit


d4e5f6g
Add login


h7i8j9k
Improve UI


z9y8x7w
Refactor auth

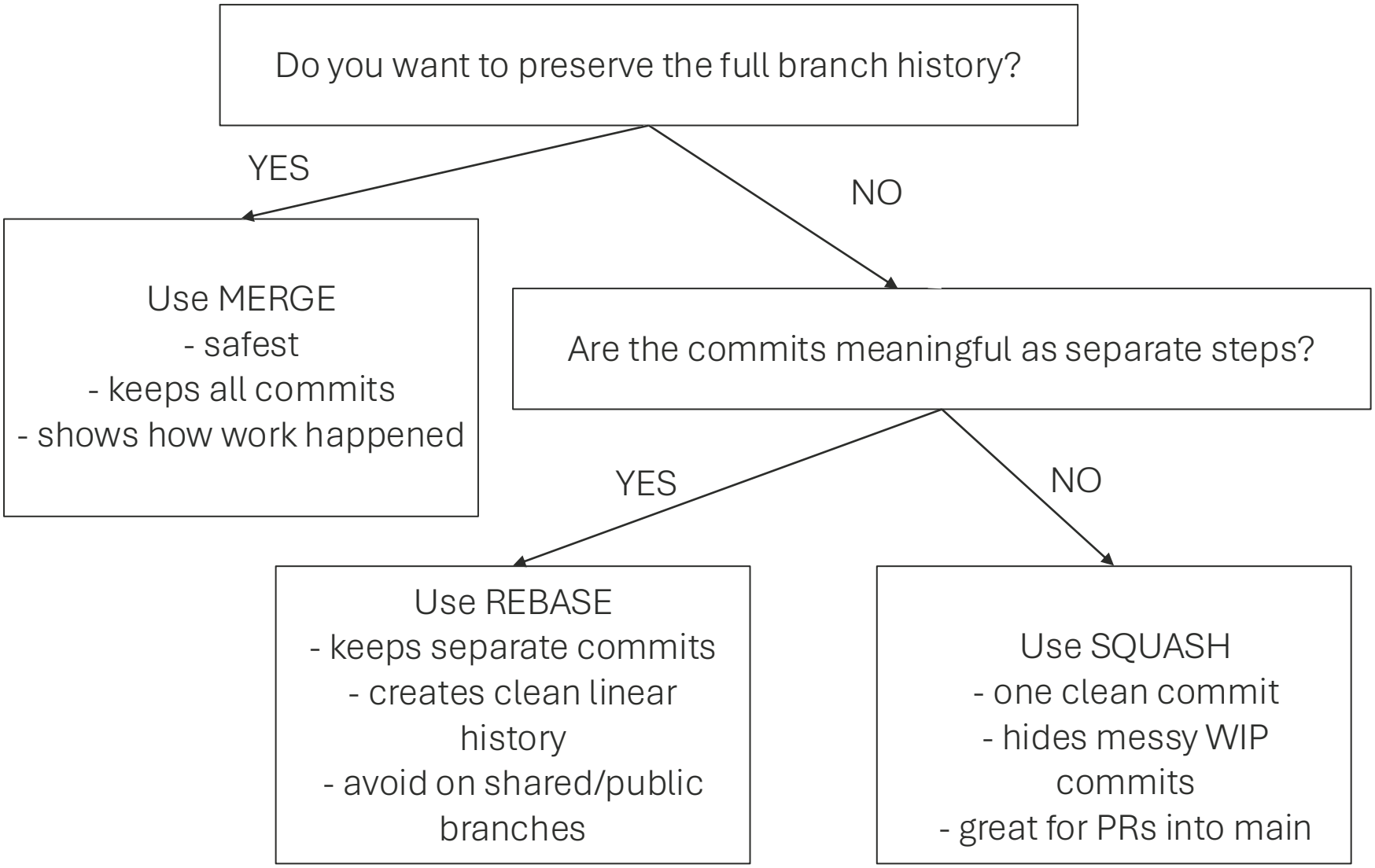
feature (squashed)


f0e1d2c
Export feature:
Add export
Fix edge case
Add tests
Update docs

All the work, one commit.

☆ Fewer commits. Clearer history. Same result. ☆

Decision making guide



End of Act 7

🎭 Act 8*

The Staging Area (why Git behaves this way)

🧠 Core idea

Working directory → your current files
Staging area → next snapshot
Repository → committed history

`git add` = update snapshot
`git commit` = save snapshot

💡 Summary

Working Dir	→ what you see
Staging Area	→ what you will commit
Commit (HEAD)	→ what you committed

End of Act 8

🎭 Act 9*

git reset (what actually happens)

```
git reset --hard
```

🧠 Core idea

- Reset = move pointers and optionally update files.
  - `--soft` -> moves the branch pointer, but keeps staging and files
  - `--mixed` -> Moves pointer and resets staging, but keeps files.
  - `--hard` -> Moves pointer, staging, and files.

- Reset doesn’t destroy commits, it just stops pointing to them.

Important nuance

- If you run garbage collection and nothing points to a commit - it ***can*** be deleted eventually.

RESET --HARD: ripping out history and starting over

End of Act 9

🎭 Act 10*

HEAD (and the scary detached one)

🧠 Core idea

- HEAD is just a pointer to your current location.
  - HEAD → branch → commit
  - OR HEAD → commit (detached)

Summary

- Detached HEAD just means:
  - Careful! You are not on a branch

A -> B -> C (main)

D (detached, no branch)

Commits don’t belong to branches.
Branches point to commits.

End of Act 10

🎭 Act 11*

What git commit actually does

🧠 Core idea

git commit =

1. read the staging area
2. create a tree
3. create a commit object
4. point it to the parent
5. move the current branch

💡 Summary

- git commit does not save “everything”
  It saves the staging area

Staging area
↓
Tree object
↓
Commit object
↓
Move branch pointer

End of Act 11

MiniAct 12*

Cherry-pick (stealing commits politely)

main: A → B → C
feature: ↘ D → E

Need only D

🧠 Core idea

Cherry-pick takes ONE commit...
and replays its changes somewhere else.

Cherry-pick does NOT move commits.
It creates a NEW commit.

Git copies the PATCH, not the COMMIT.

💡 Summary

Cherry-pick = replay ONE commit

✅ great for:

- hotfixes
- backports
- wrong-branch mistakes

⚠️ creates NEW commits

⚠️ duplicates history intentionally

Imagine commits are pages in a book.

Cherry-pick is not
ripping out the original page.

It is photocopying the page...

...and inserting the copy
into another book.

End of MiniAct 12

🎭 Bonus Mini Act

Git learns parallel universes (Worktrees)

🧠 Core idea

Git separates:
🧱 object database
from
📂 working directories

Worktrees prove this.

Working directory != repository

💡 Summary

Worktrees allow:

- ✅ multiple active branches
- ✅ shared Git storage
- ✅ isolated working directories

Perfect for:

- 🤖 AI agents
- 🧪 experiments
- 🔥 hotfixes
- ⚡ parallel work

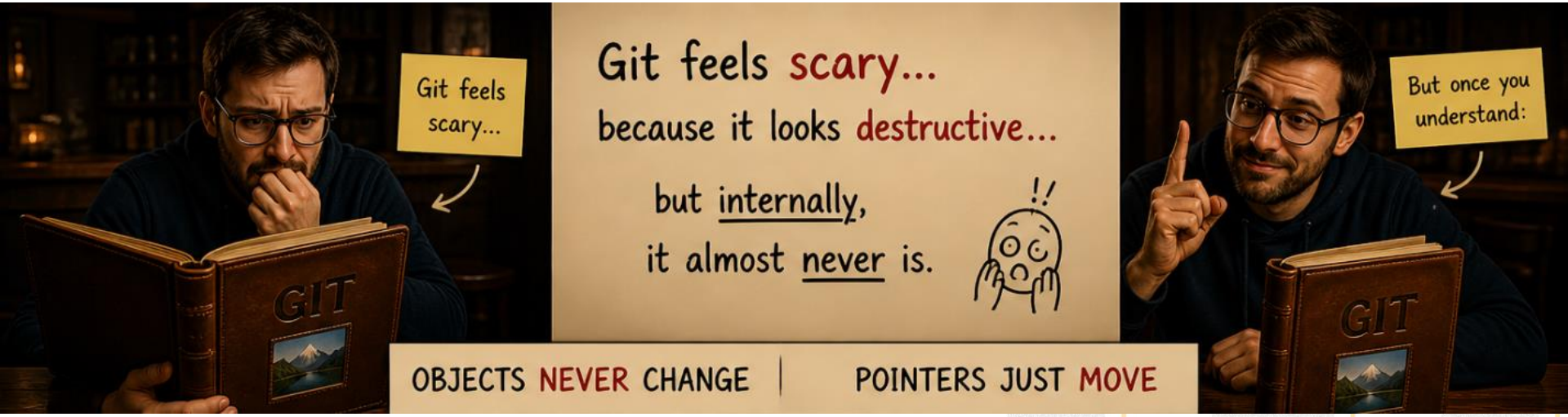
End of bonus miniact

Closing act

Why Git feels scary (and why it shouldn’t)

🧠 Core idea

Git feels scary... because it *looks* destructive...
but internally, it almost never is.



1.  It looks like things disappear

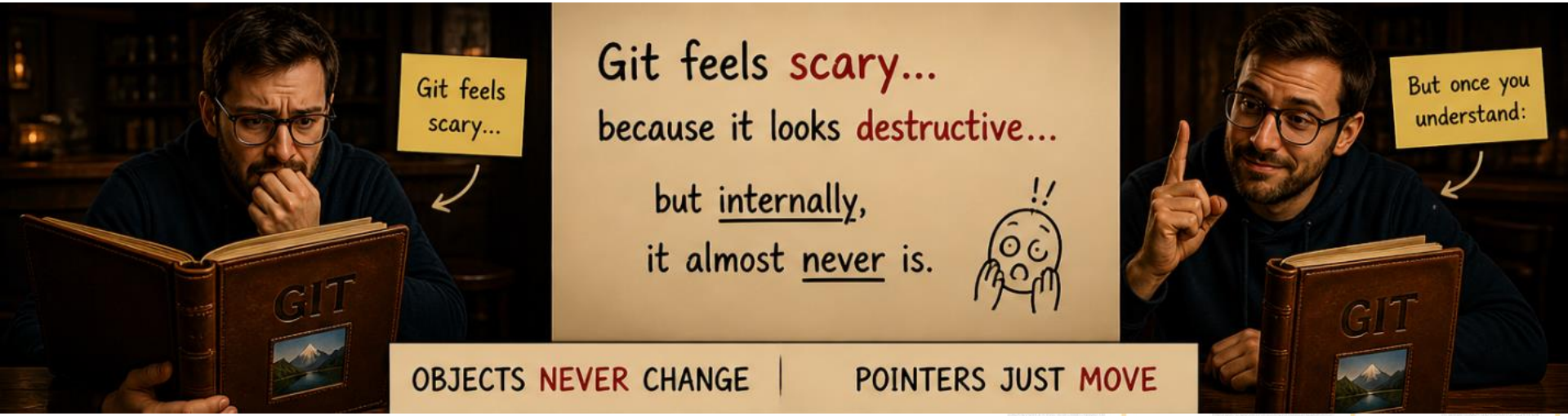
- `reset --hard`
- detached Head
- rebase

2. 🧠 The commands are misleading

- `git add` → doesn't "add to repo"
- `git reset` → doesn't "reset everything"
- `git checkout` → does too many things

A bit of honesty at the end

- 🧨 Git doesn't *always* save you...

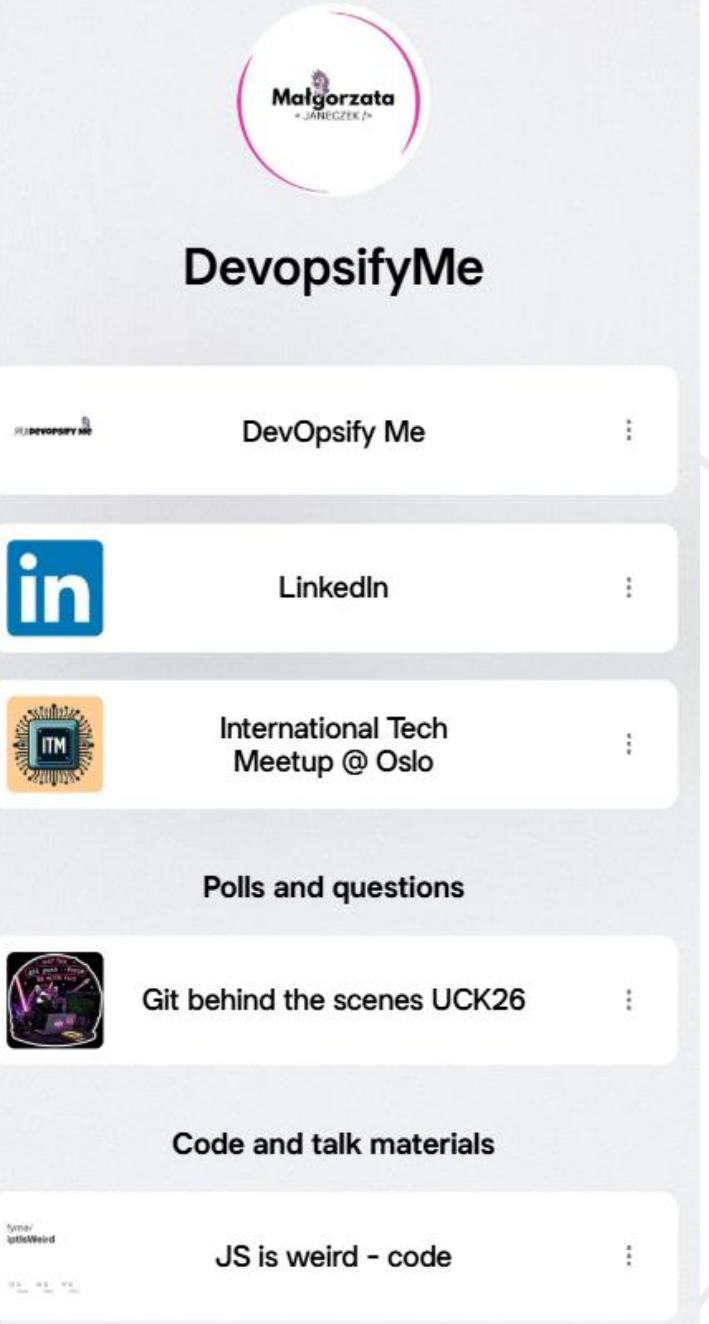


⚠ You probably won't use these commands

Some of the commands that will be shown today are low-level Git internals (plumbing commands)

- `git commit = git write-tree, git commit-tree`
- `git add = git update-index`
- `git show = git cat-file`

Self-promotion linktree



Update Conference
Krakow 2026

Thank you

QnA o’clock!