Git-ting started

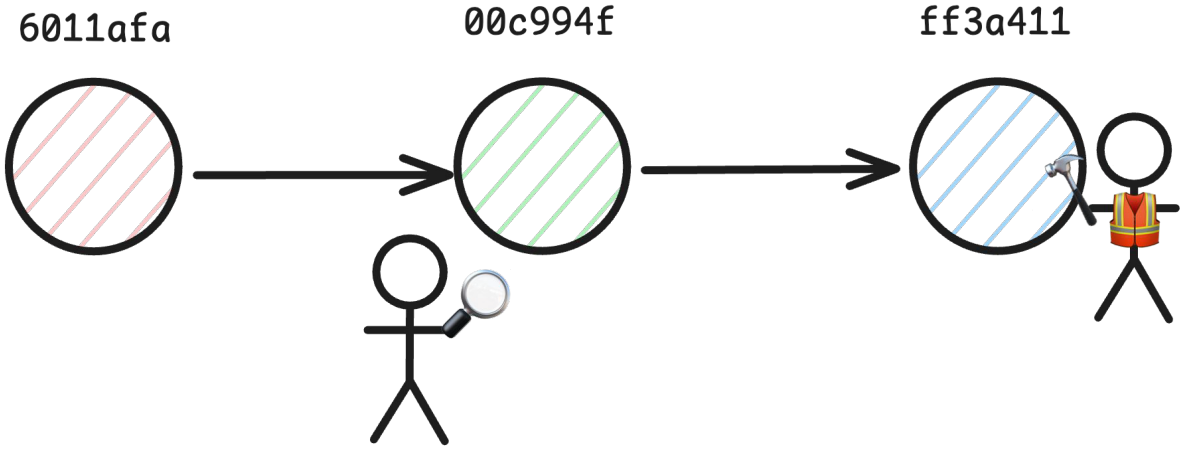
A small introduction to git

Agenda

- Motivation behind the talk
- Demonstration
- Usage circumstances
- FAQ

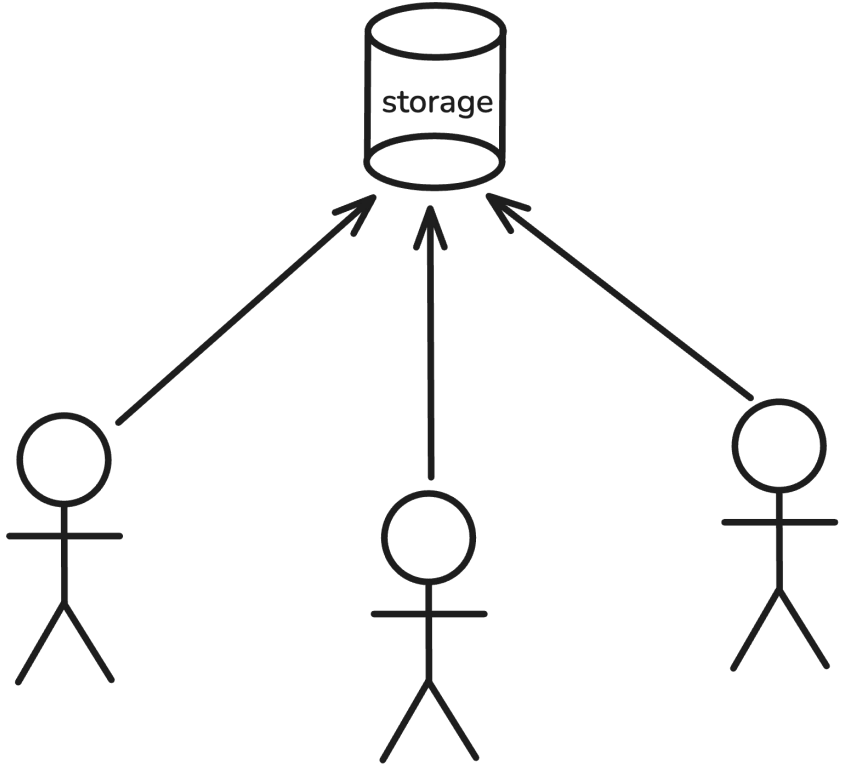
Versioning

- A label to different stages of a project
- Structured changes



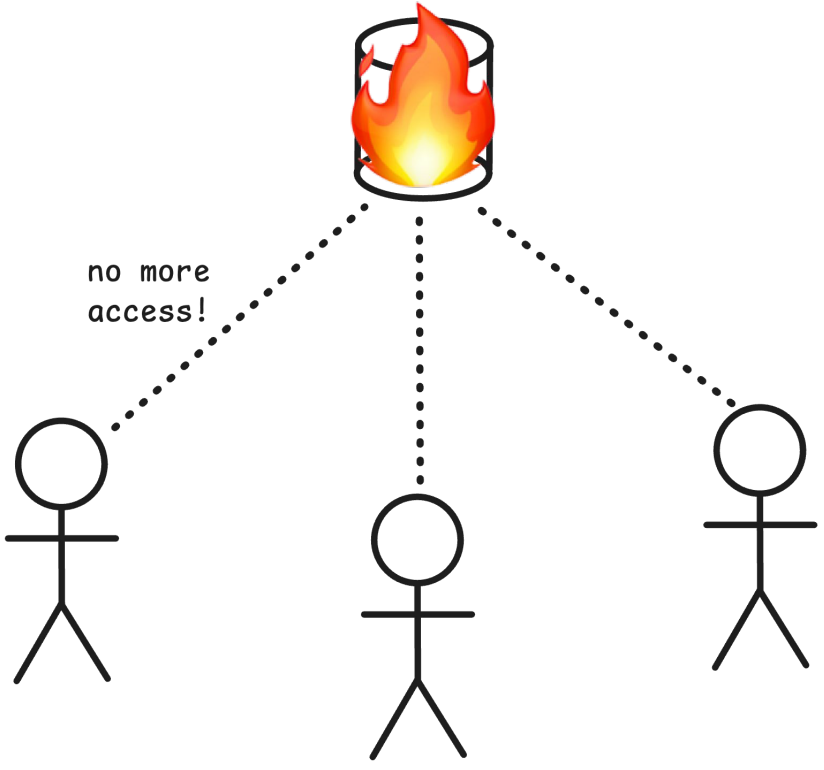
A history of version systems

- Local revision systems not so helpful in collaboration
- Why not make this a *centralised* version control system (VCS)



A history of version systems

- Centralised servers can **break**

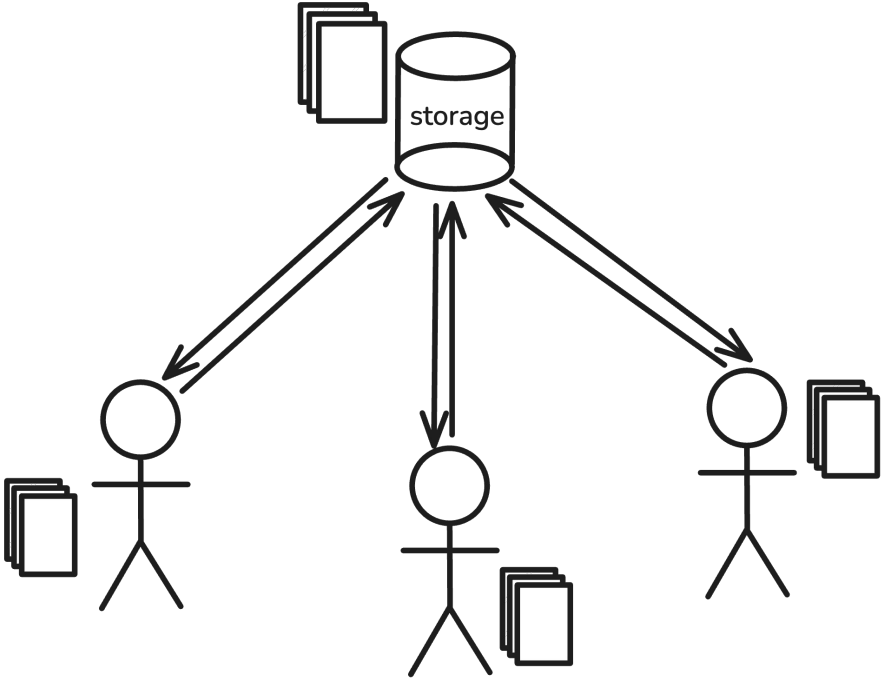


Fast forward to 2005

- Torvalds introduces **Git**
- A *distributed* VCS

What's so great?

- Everyone has their own copy
- Changes are local
- Literally **time-travel** through your *committed* changes
- Switch between different project *branches*

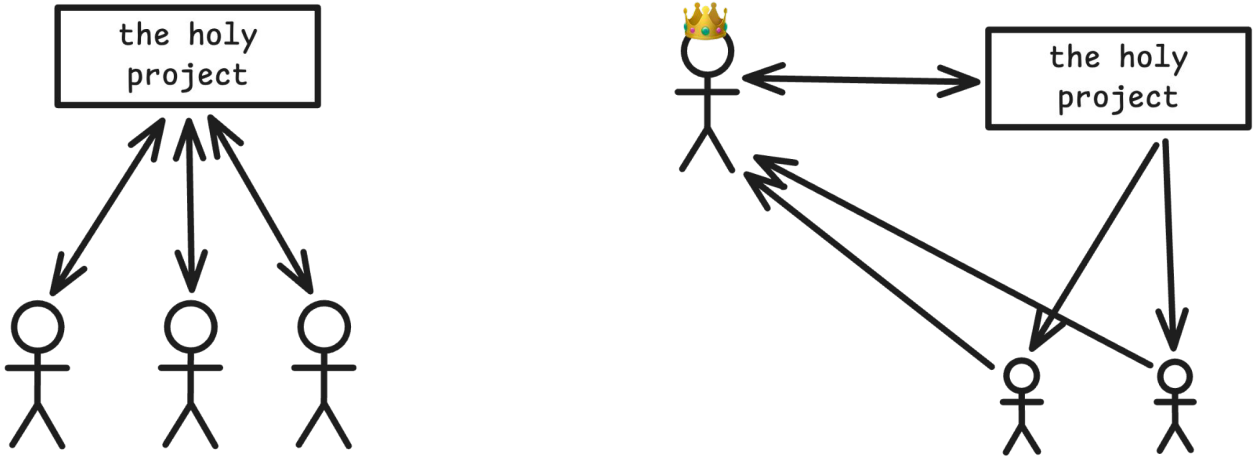


(Originally developed for the Linux kernel, so yes it works at a large scale)

It’s also Free & Open Source Software

Power of a distributed VCS

- Supports ANY sort of workflow



Overview of git

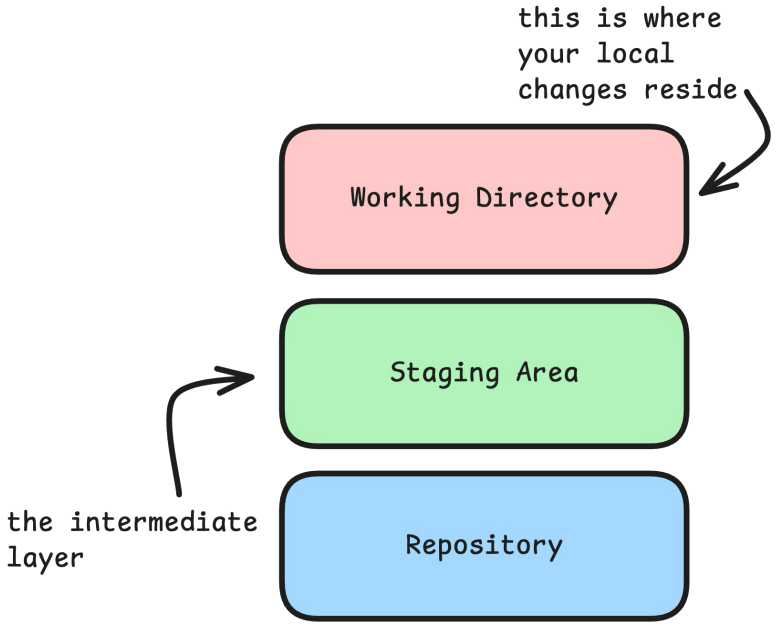
Some common terms

- Repository: A place where git stores all of its tracked objects, this is where your project *lives.*
- Commit: A point in the git history.
- Branches are a *line of development*.
- head is the tip commit of the branch, **HEAD** is the current branch
- Working tree is the HEAD's contents and any *uncommitted* changes

source: git-scm

The staging area

- Your local changes aren't really a part of the repository
- There are *layers*



source: git-scm

It’s basically this key-value *database*

Like literally

- **Git objects** are the things that store all the data
- Generates a unique key for a data object

```
[grogu:~/foo]$ echo "tilde" | git hash-object -w --stdin
23aef7f12e588e175eba3ee170a8341414a1ac62

[grogu:~/foo]$ git cat-file -p \
    "23aef7f12e588e175eba3ee170a8341414a1ac62"
tilde
```

Where are these files stored?

- Git creates a hidden .git directory with a bunch of things in it
- Created with initialization

Let's begin with `init`

Initialization

- You don’t have to manually create the `.git` file
- Just run `git init` within your project directory

```
[grogu:~/foo]$ git init
Initialized empty Git repository in /home/grogu/foo/.git/
[grogu:~/foo]$ ls -la
total 0
drwxr-xr-x 1 grogu grogu   8 Jul  7 10:43 .
drwx------ 1 grogu grogu 402 Jul  7 10:46 ..
drwxr-xr-x 1 grogu grogu  82 Jul  7 10:43 .git
```

We have now created an empty git repository

Inspecting new repo

- The status command gives us an overview of the current working tree
- Shows paths different from working tree and index files
- Shows paths of untracked files and the branch

```
[grogu:~/foo]$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

A fresh repo has nothing for us to see

Let’s get things to be tracked

```
[grogu:~/foo]$ echo "# gitting started" > donuts.md
[grogu:~/foo]$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be
committed)
        donuts.md

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

Created a fresh file with text inside it

Let’s create a commit

- The add command prepares the content into the staging area
- Updates the index
- A commit will save the staged region to the repository
- It’s essentially a snapshot of the repository with the updated index

```
[grogu:~/foo]$ git add donuts.md
[grogu:~/foo]$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   donuts.md
```

Donuts here is now ready to be committed

Prerequisite

- It’s better to set a global set a global email and name before committing
- The commit will be associated with this email and name

```
git config --global user.email "adityahegde.clg@gmail.com"
git config --global user.name "Aditya Hegde"
```

- Or edit the `~/.gitconfig` file manually

```
[init]
        defaultBranch = main
[user]
        email = adityahegde.clg@gmail.com
        name = Aditya Hegde
```

Writing a commit

- With the files staged, we can now create a commit ✨
- You *must* specify a message associated with the following commit

```
[grogu:~/foo]$ git commit -m "genesis"
[main (root-commit) 0be7a3a] genesis
 1 file changed, 1 insertion(+)
 create mode 100644 donuts.md
```

You now have your first commit 🥳

The *logs*

- View the reachable commits from a parent

```
[grogu:~/foo]$ git log
commit 0be7a3a50ba3e2dbd8b3df11549d8f869db4f13f (HEAD →
main)
Author: Aditya Hegde <adityahegde.clg@gmail.com>
Date:   Mon Jul 7 11:24:57 2025 +0530

    genesis
```

- Or if your into a minimal sort of view

```
[grogu:~/foo]$ git log --oneline --decorate --graph --abbrev
* 0be7a3a (HEAD → main) genesis
```

Making a few changes

- What if we modify the repository

```
[grogu:~/foo]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:   donuts.md

no changes added to commit (use "git add" and/or "git commit -a")
```

- Status shows us the modified paths

A more detailed view?

- A diff view lets you see the per line modifications
- Changes between the working tree and the index

```
diff --git a/donuts.md b/donuts.md
index cebc788..354b600 100644
--- a/donuts.md
+++ b/donuts.md
@@ -1 +1,3 @@
-# gitting started
+# Gitting Started
+
+> An introduction to git!
```

- Red: Modified/deleted line in the index
- Green: What has been updated in place in the worktree

A more detailed view?

- Lots of editors show these *signs* of modification in the text buffer

```
~   1 # Gitting Started
+   2
+   3 > An introduction to git!
```

An example of how vim does this with git-gutter

Multiple commits

- With another commit, lets see how the commit logs change

```
commit b8c376474e1e31d2416322d08f62e89d643e2a03 (HEAD → main)
Author: Aditya Hegde <adityahegde.clg@gmail.com>
Date:   Mon Jul 7 13:44:45 2025 +0530

    chore: update donuts.md

commit 0be7a3a50ba3e2dbd8b3df11549d8f869db4f13f
Author: Aditya Hegde <adityahegde.clg@gmail.com>
Date:   Mon Jul 7 11:24:57 2025 +0530

    genesis
```

- The **HEAD** now points to the latest commit b8c3764

Local → Remote

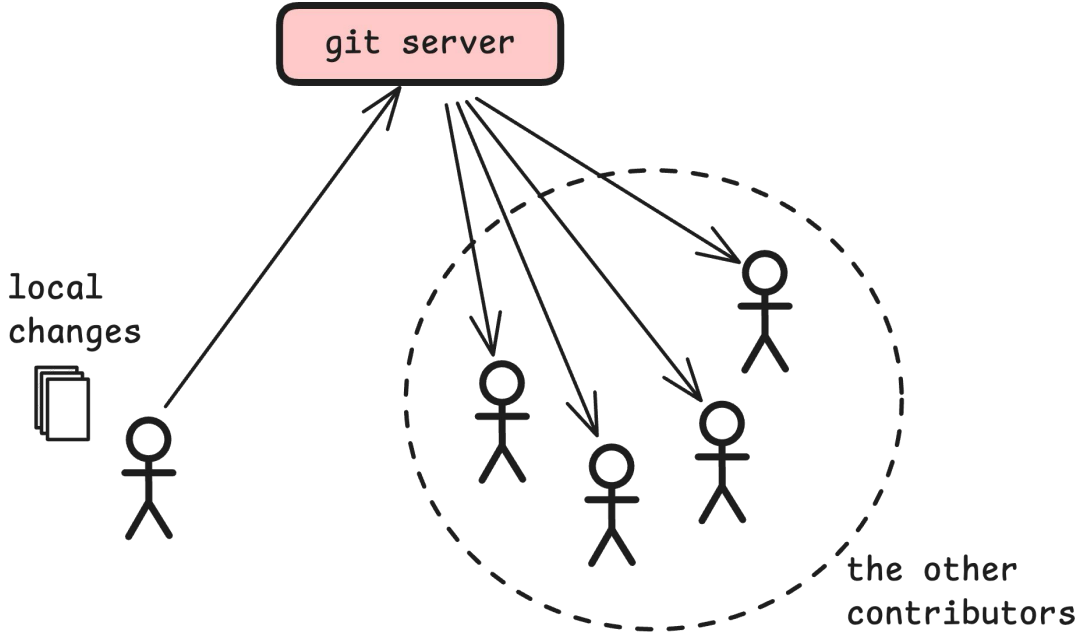
Pushing changes to a *remote* repository

- We’ve had our changes locally so far.
- But what if others want to *pull* our changes/files.
- We need to push then changes to some *remote* server where everything is stored.

But where & *how*??

Adding a remote repository

- Need of a remote server where repo can be stored
- A common place from where others can *clone* the repo



Adding a remote repository

- Let's assume we already have a server

```
[grogu:~/foo]$ git remote add origin \
        grogu@server.orb.local:/home/grogu/foo.git
```

- To break that above command down
  - **`grogu@server.orb.local`** is the server at which the repository is stored
  - **`/home/grogu/foo.git`** is the path where the repository is located

- This might seem very familiar if you have seen the git remote address (ex: `git@github.com:bwaklog/foo.git`)

Checkout git-scm for more on `remote`

Push it :P

- Lets now push our changes to the remote server labeled origin

```
[grogu:~/foo]$ git push origin -u main
Enter passphrase for key '/home/grogu/.ssh/id_ed25519':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 505 bytes | 505.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To server.orb.local:/home/grogu/foo.git
 * [new branch]      main → main
branch 'main' set up to track 'origin/main'.
```

Your commit is now pushed to the server 🥳

Don’t believe it?

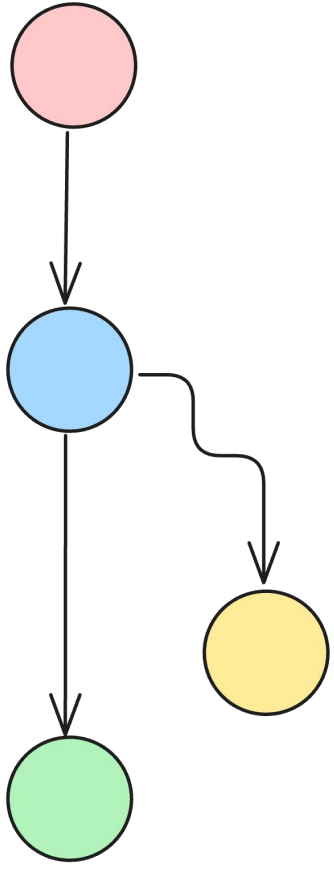
```
[bar@foo ~]$ git clone grogu@server.orb.local:/home/grogu/foo.git
Cloning into 'foo'...
Enter passphrase for key '/home/bar/.ssh/id_ed25519':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from
0)
Receiving objects: 100% (6/6), done.
```

That’s another VM, don’t believe me?

```
[bar@foo foo]$ git log --oneline --decorate
b8c3764 (HEAD → main, origin/main, origin/HEAD) chore: update
donuts.md
0be7a3a genesis
```

Same commit history :D

Branches



What's a *branch*

- A different line of development on the repo
- Make changes, but without disturbing the main line
- `checkout` lets us switch between them or restore working tree files

```
[grogu:~/foo]$ git checkout -b feat/script
Switched to a new branch 'feat/script'
```

Branches

- Let’s make a small commit on the new branch

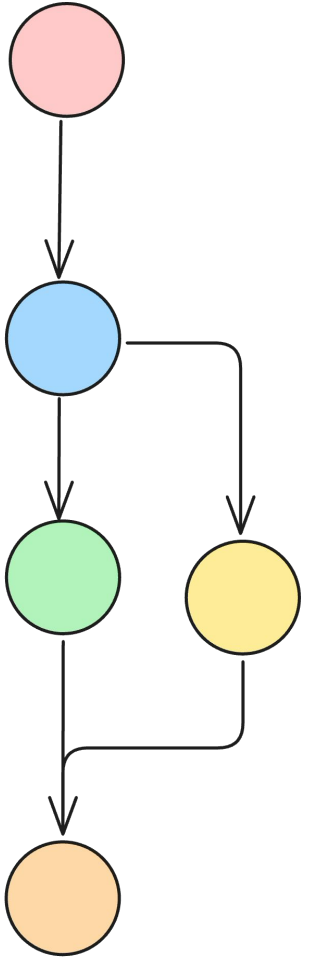
```
* 544f594 (HEAD → feat/script) feat: added a random script
* b8c3764 (origin/main, main) chore: update donuts.md
* 0be7a3a genesis
```

- Well, how does the main branch look?

```
* b8c3764 (HEAD → main, origin/main) chore: update donuts.md
* 0be7a3a genesis
```

Still the same `:D`

Merging branches



What's a merge

- Well…

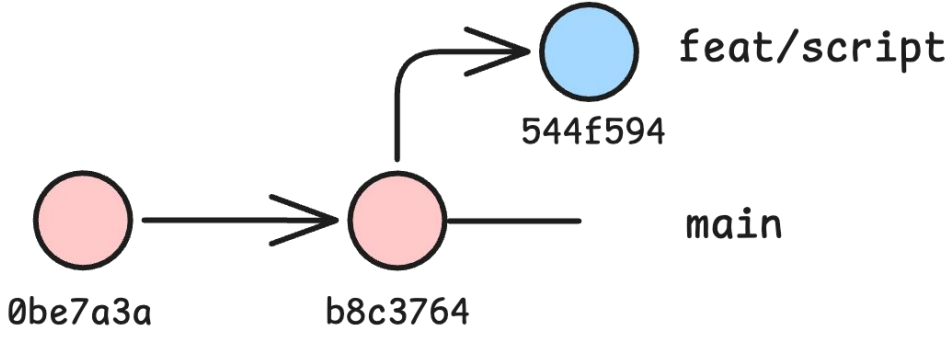
"combine or cause to combine to form a single entity"

- But with respect to Git…

"Join two or more development histories together"

- Incorporating changes from named commits into the *current branch.*

Merging branches



What we are currently working with

Merging branches

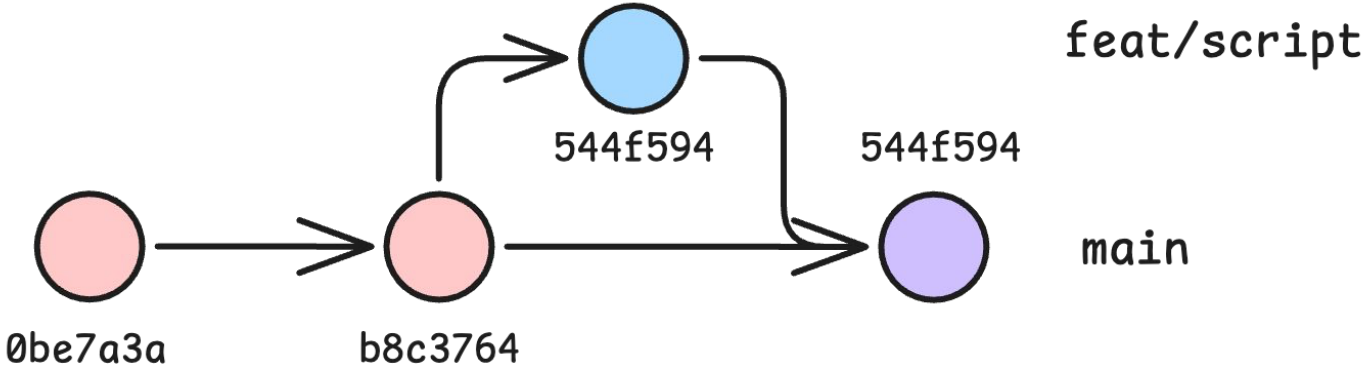
```
[grogu:~/foo]$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Make sure you are on the main branch

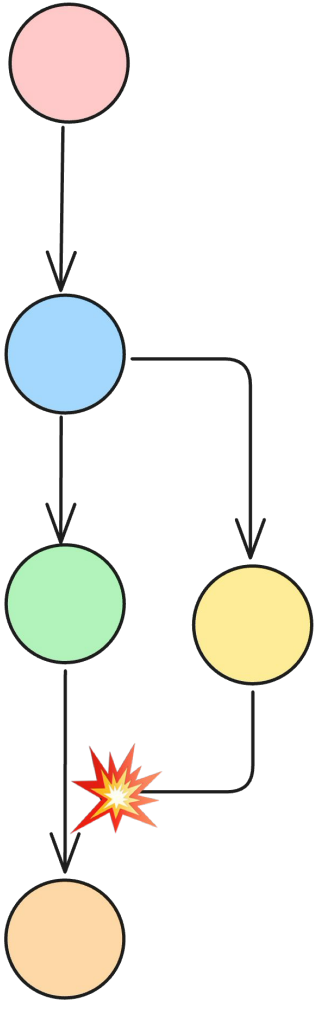
```
[grogu:~/foo]$ git merge feat/script
Updating b8c3764..544f594
Fast-forward
 donuts.md | 2 ++
 script.py | 4 ++++
 2 files changed, 6 insertions(+)
 create mode 100644 script.py
```

Merging branches

- Git replays the changes made since `feat/script` diverged from the main branch
- Results in the new commit on the main branch
- What if it can’t resolve this??



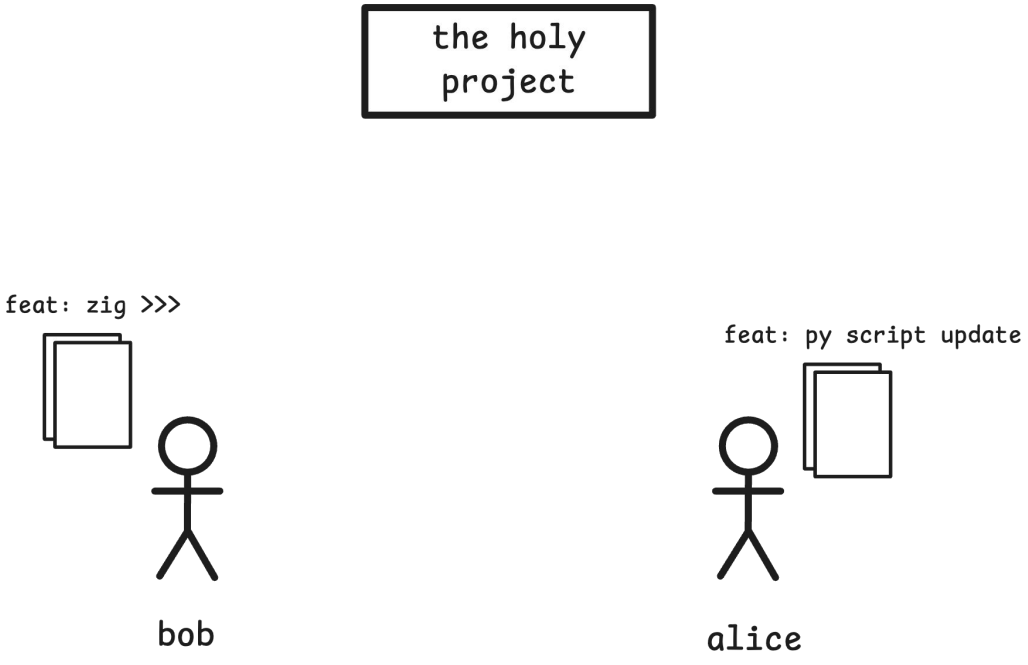
Conflict?!



Conflicts?!

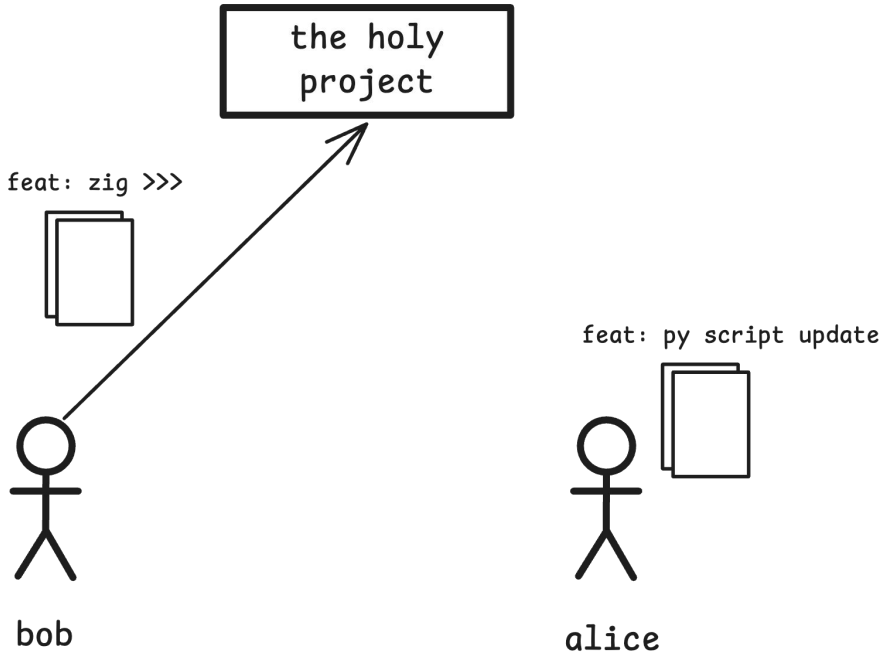
- Sometimes Git just can’t help your differing changes
- When two different branches/clones do opposing changes?

Conflicts?!



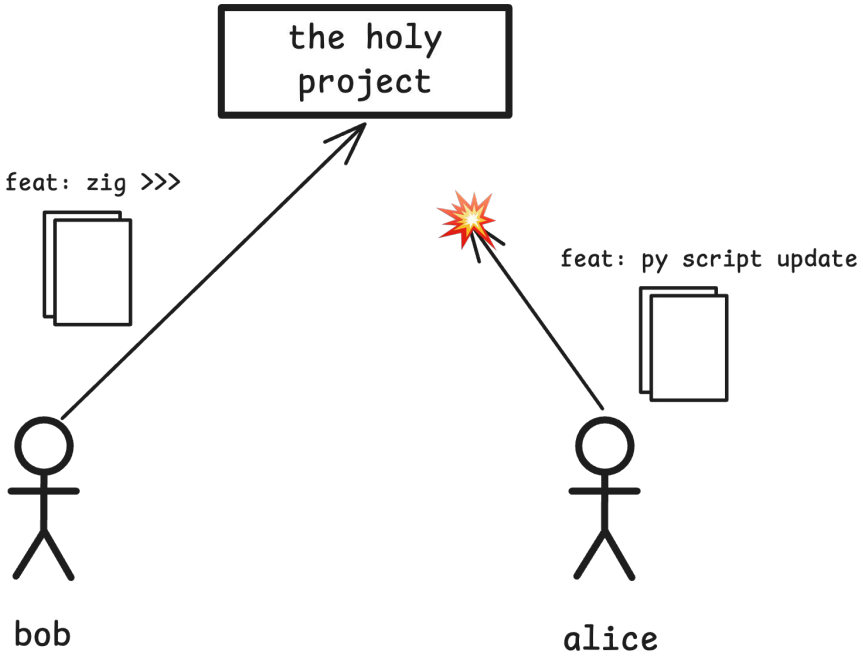
Alice and Bob make two different commits on their local branches

Conflicts?!



Bob pushes his changes before alice

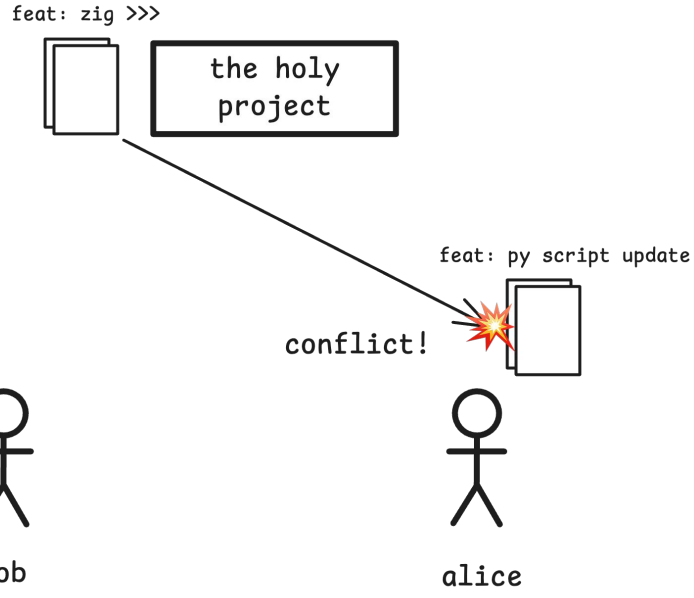
Conflicts?!



Alice can’t push after Bob, Alice pulls the changes

Voila! A conflict

- We finally have a *very* possible scenario



Conflicts

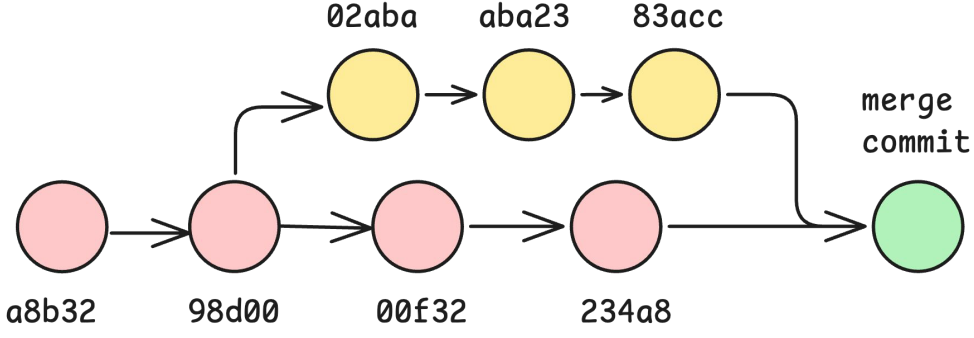
- Alice might see something like this
- You might come across things like *rebase*, *no-rebase* and *fast forward*, what is this?

```
[bar@foo foo]$ git pull origin
...
...
From server.orb.local:/home/grogu/foo
   544f594..8c167e1  main       → origin/main
hint: You have divergent branches and need to specify how to
reconcile them.
hint: You can do so by running one of the following commands
sometime before
hint: your next pull:
...
```

Merge vs rebase vs reset

Merge to solve conflicts

- We can solve with merges
- Manual work on choosing *incoming* and current changes



No rebase

- The way to solve conflicts with merges
- Can handle for simple modifications
- Others need to be solved manually
- You can always abort a merge with `git merge --abort`

```
[bar@foo foo]$ git pull origin --no-rebase
Enter passphrase for key '/home/bar/.ssh/id_ed25519':
Auto-merging donuts.md
CONFLICT (content): Merge conflict in donuts.md
CONFLICT (modify/delete): script.py deleted in
8c167e1ea00eed2ade247f6ba7b2e26e15a8997a and modified in HEAD.
Version HEAD of script.py left in tree.
Automatic merge failed; fix conflicts and then commit the result.
```

Unresolved merge files

- Here's an example

```
+   4 <<<<<<< HEAD
+   5 - Contains a script `script.py`
+   6 =======
+   7 - Zig really cool
+   8 >>>>>>> 8c167e1ea00eed2ade247f6ba7b2e26e15a8997a
```

- The **<** bracket line is your current branch
- The **>** bracket line is other branch we are merging from
- The **=** divider splits the conflicting changes

How to solve that mess

- Manually editing those signs
- Some editors come with **merge conflict** editors (VSCode, IDEs, Zed, …)
- You control how the resolved file looks

```
+   1 # Gitting Started
+   2
+   3 > An introduction to git!
+   4
+   5 - Zig really cool
+   6 - And, there's a script `script.py`
```

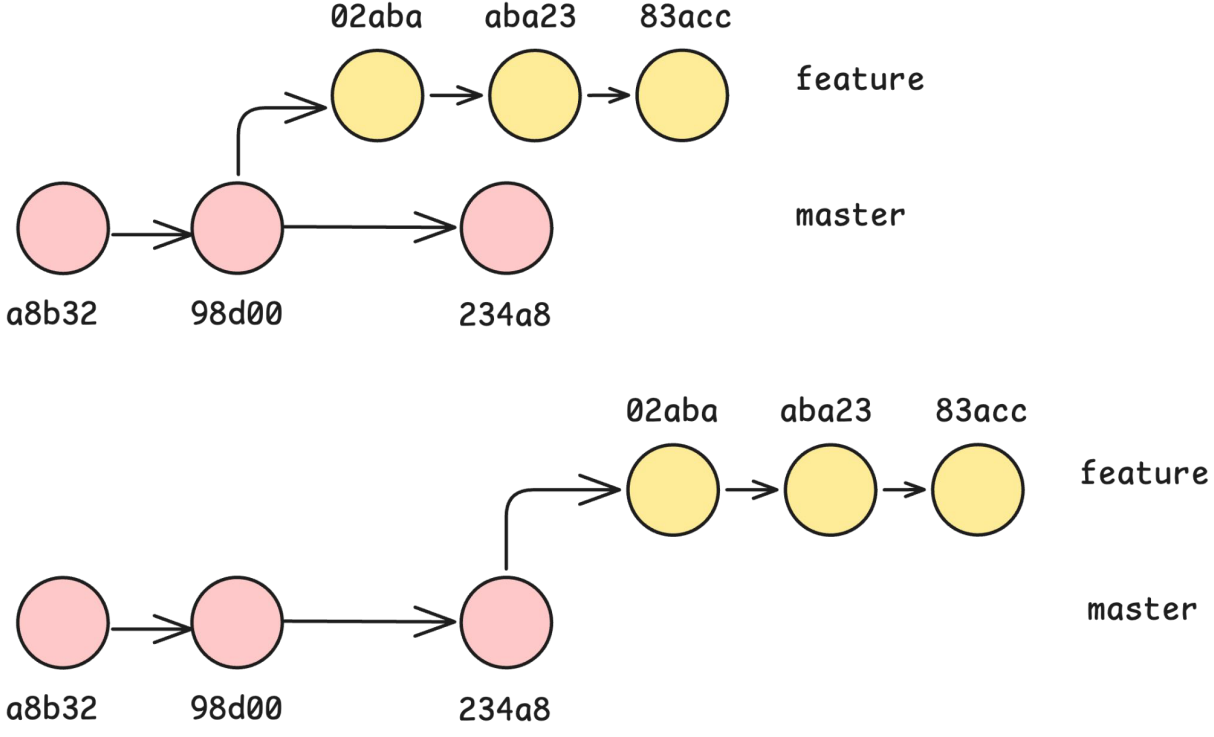
Resolved conflicts

How to solve that mess

- The merge will be a commit itself
- You control what files remain at the end
- Once solved we `commit` to create our final *merged commit*

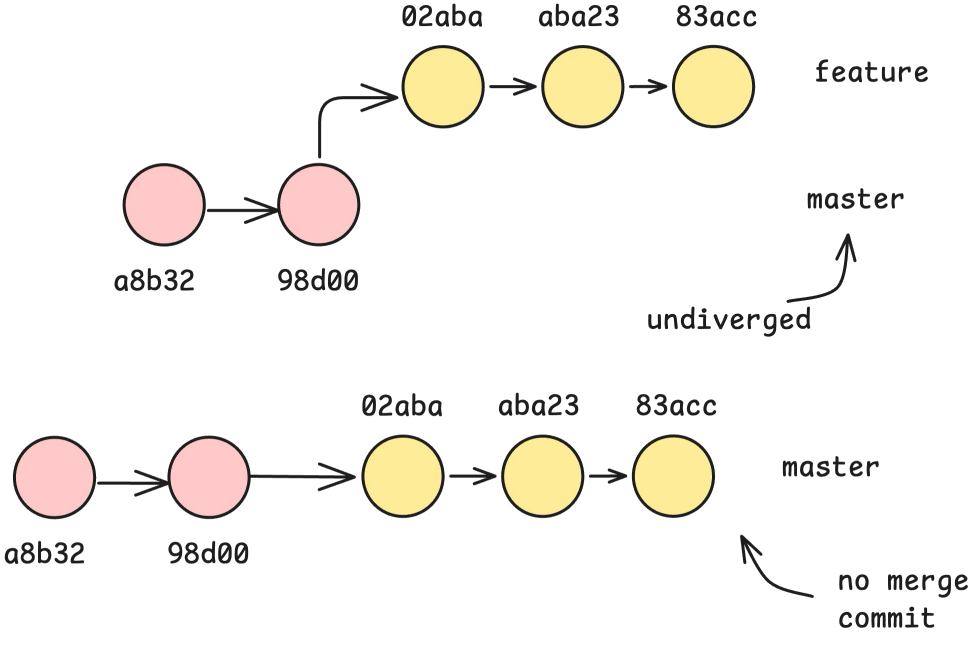
Rebase?

- Literally moving a sequence of commits to occur upon a different base



Fast forward

- Only possible if master is un-diverged
- Git points the commit to the latest commit on the feature branch



Manipulating Commits

Forgot a file? Typo in commit message?

- Attach **`--amend`** to rewrite your most recent commit msg, without creating a new one

```
$ git add missed_file.txt
$ git commit --amend
```

Manipulating previous *commits*

- Git rebase is your friend
- With great power comes great responsibilities :P
- You can change messages, reorder, delete or even squash commits!

```
pick b8c3764 # chore: update donuts.md
pick 544f594 # feat: added a random script
pick 75aa674 # feat: py script update
pick 8c167e1 # feat: zig >>>

# Rebase 0be7a3a..8768035 onto 0be7a3a (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# ...
```

(Accidentally) micro-committed 34 times?

- Git lets us take multiple commits and **squash** them into a single one

Ignore or Hiding things

Ignoring things

- Some things which have *private data*
- Subfolders with gigabytes of data, can’t commit each time
- The .gitignore files

Stash 

Save work, without committing :P

Sometimes you're halfway through a feature, but need to pull in updates

Stash your changes!

- Temporarily saves changes, that you can come back to later
- Changes disappear from working tree, but not forever
- Include ex of git stash, list, apply, drop, pop (or just stash list and pop)

More on stashing and cleaning

And finally....*Github*

FAQ

Squashing?

- No problemo, just **squash** your commits!

```
$ git rebase -i HEAD~3
```

- Pick one commit, squash the rest

```
pick  hgs6sa  add signup page
squash sjhd7a fix typo
squash 78sha7 align buttons
```

- Note that the resulting commit looks like:

```
commit 892h2a
+ Add signup page, fix typo, align buttons
```

Patching 

Share just the changes 😌

- Instead of pushing whole branch/ repo, just the diff bw two/more commits
- Self contained, shareable diffs
- Generate patch file
- file.patch
- Maybe insert patch file format
- Great for mailing lists/ code reviews!