

# GIT GUD

# **Agenda**

1. Background
2. Internals
3. Tips & Tricks
4. Best Practices

1. => Lecture
2. => Lecture & Interactive
3. => Interactive
4. => Lecture

# Background

**what is Git?**

Ask this question?

**VCS**

**Version Control System**

**What does a VCS do?**

[...] a system that records changes to a file or set of files over time

– **Scott Chacon & Ben Straub** ProGit

# **Version Control Systems**

- CVS
- Perforce
- SVN
- Mercurial
- Git

Who has tried what?  
Later: How they differ

# **Open Source**

**GNU GPL2**

published under GNU General  
Public License version 2

^

Next Slide: What makes Git  
different from, for example,  
SVN

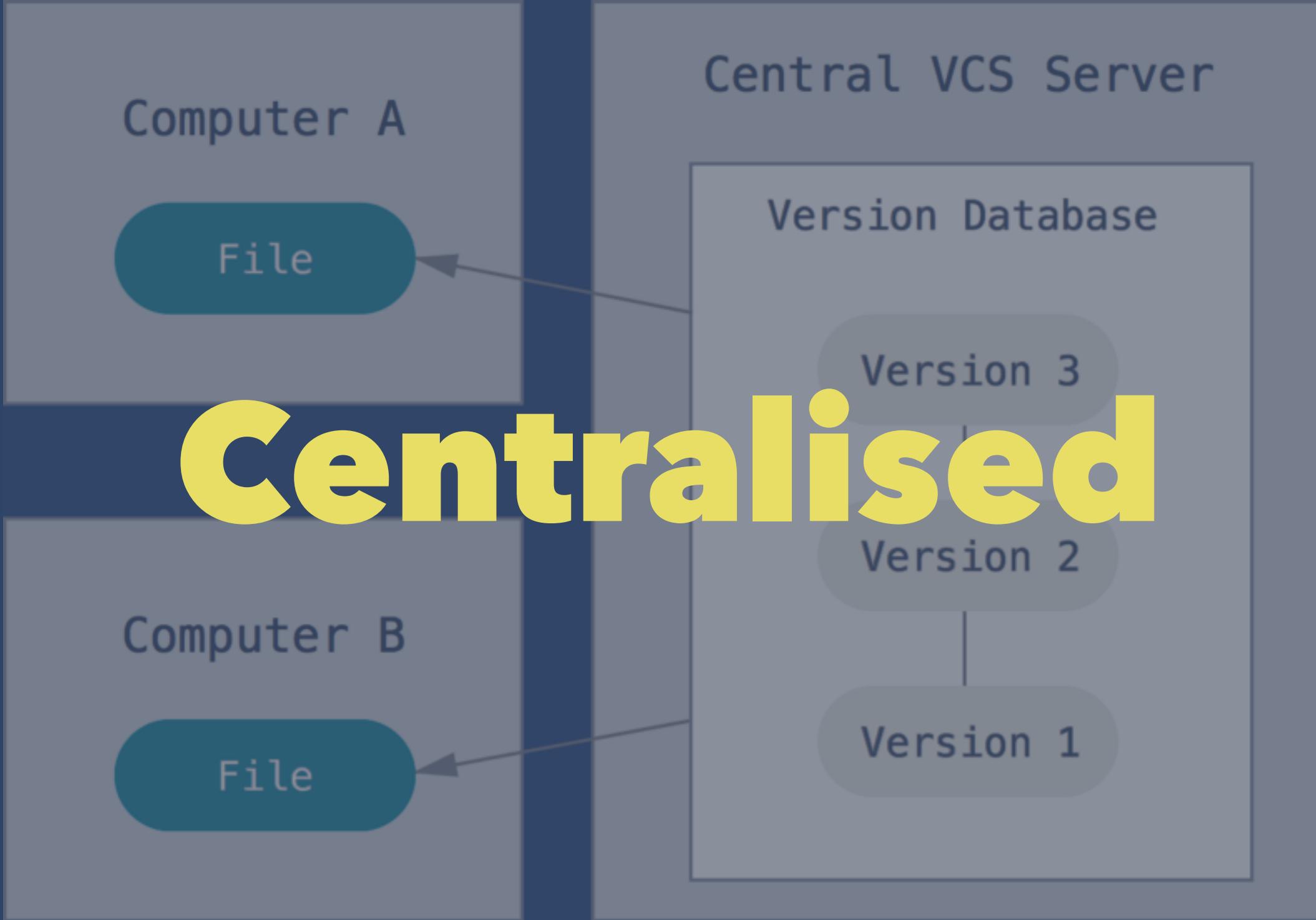
# **Distributed**

What does that mean? For this we have understand:

# Centralised

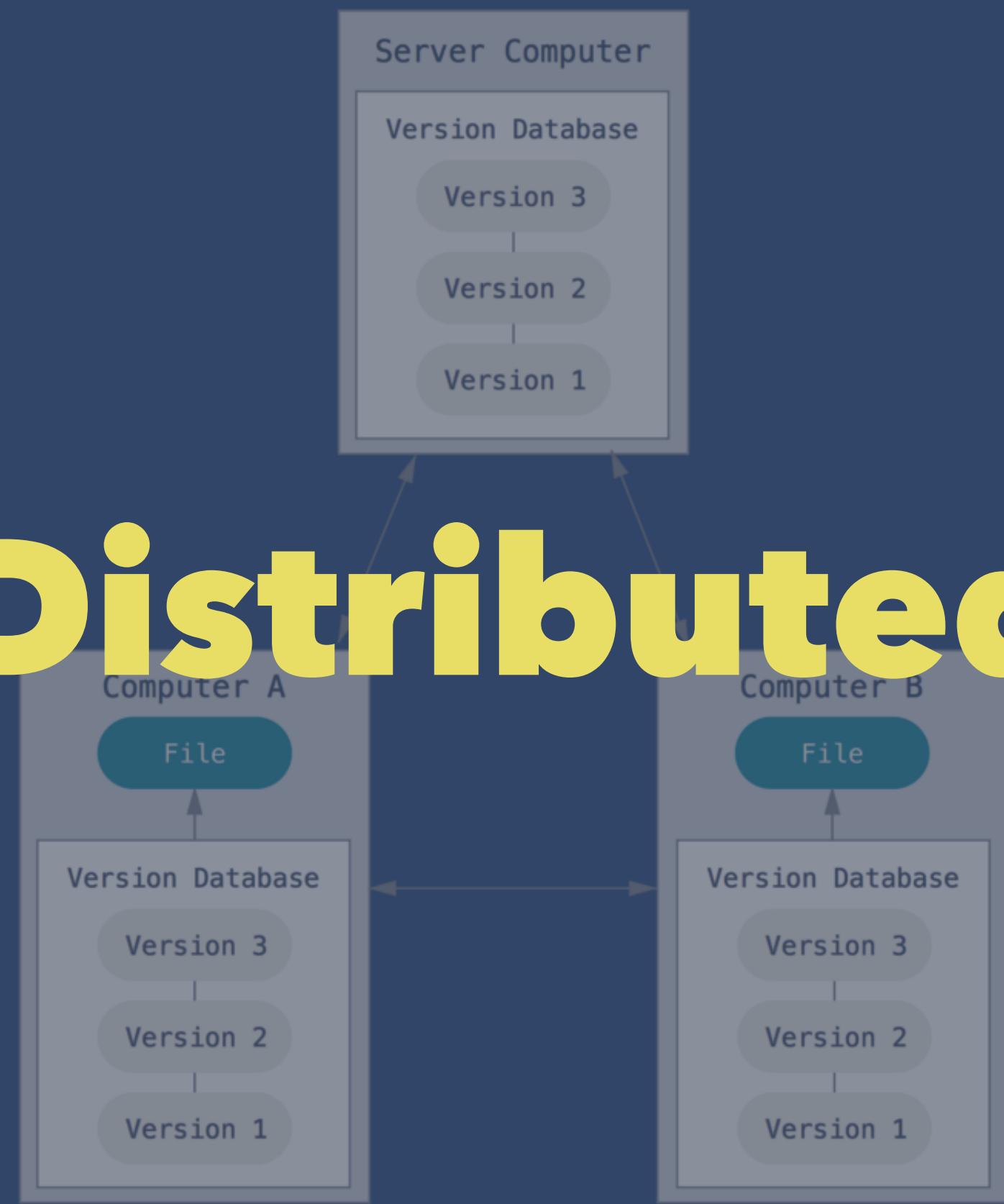
vs

# Distributed



- Central repository = Single source of truth
  - Server/Client communication
- Examples: SVN, CVS, Perforce

# Distributed



- Each repository a full backup (with exceptions)
- Communication between repositories
- Possible to do centralised approach (usual approach)

Examples: Git, Mercurial



# Internals

let's get our hands dirty

I could bore with commands, but you probably know those already. Let's talk about how Git works internally => Helps a lot for advanced stuff

- Terminology
- Object model

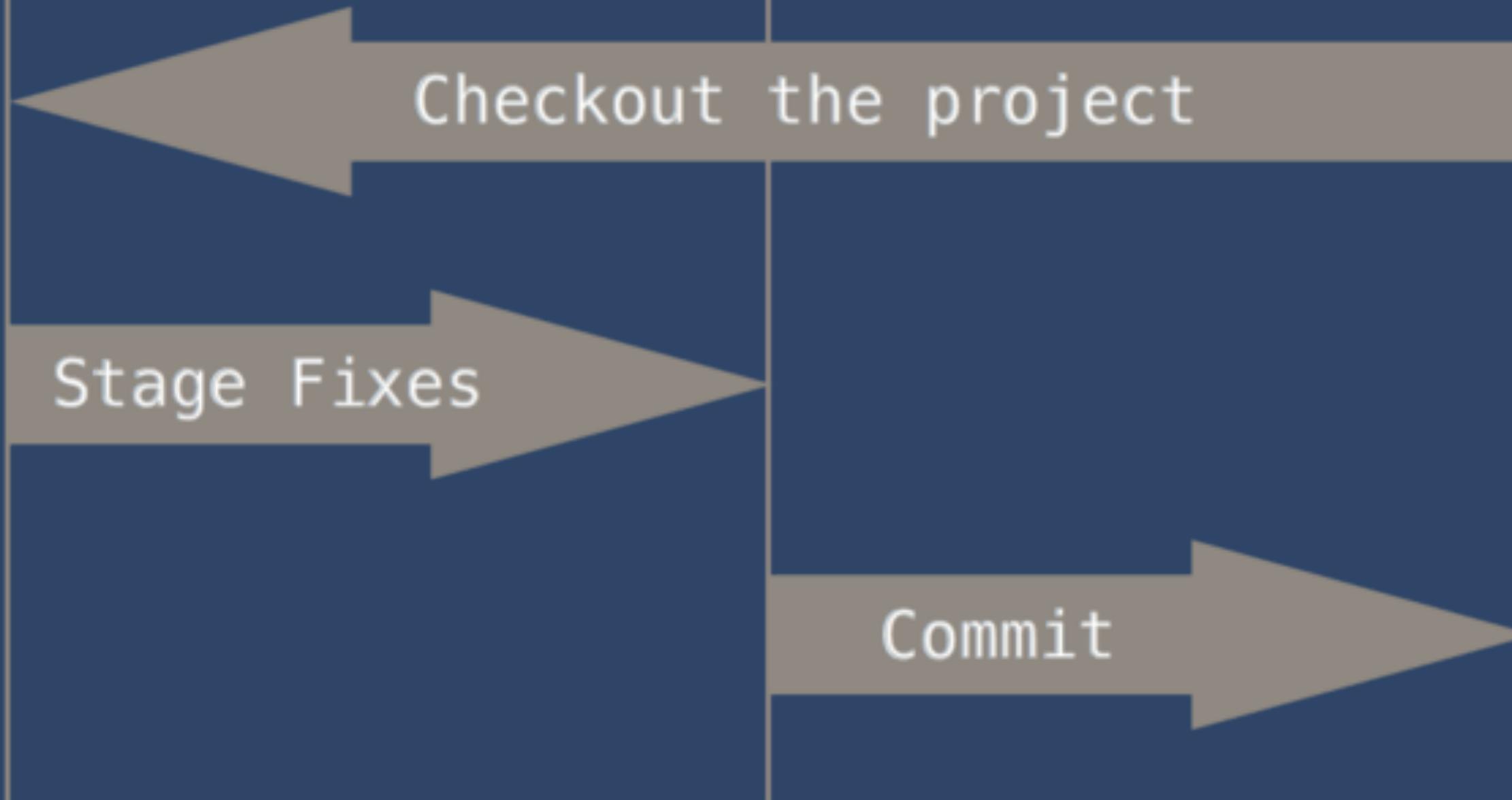
# Expected Knowledge

- add
- reset
- commit
- merge

Working  
Directory

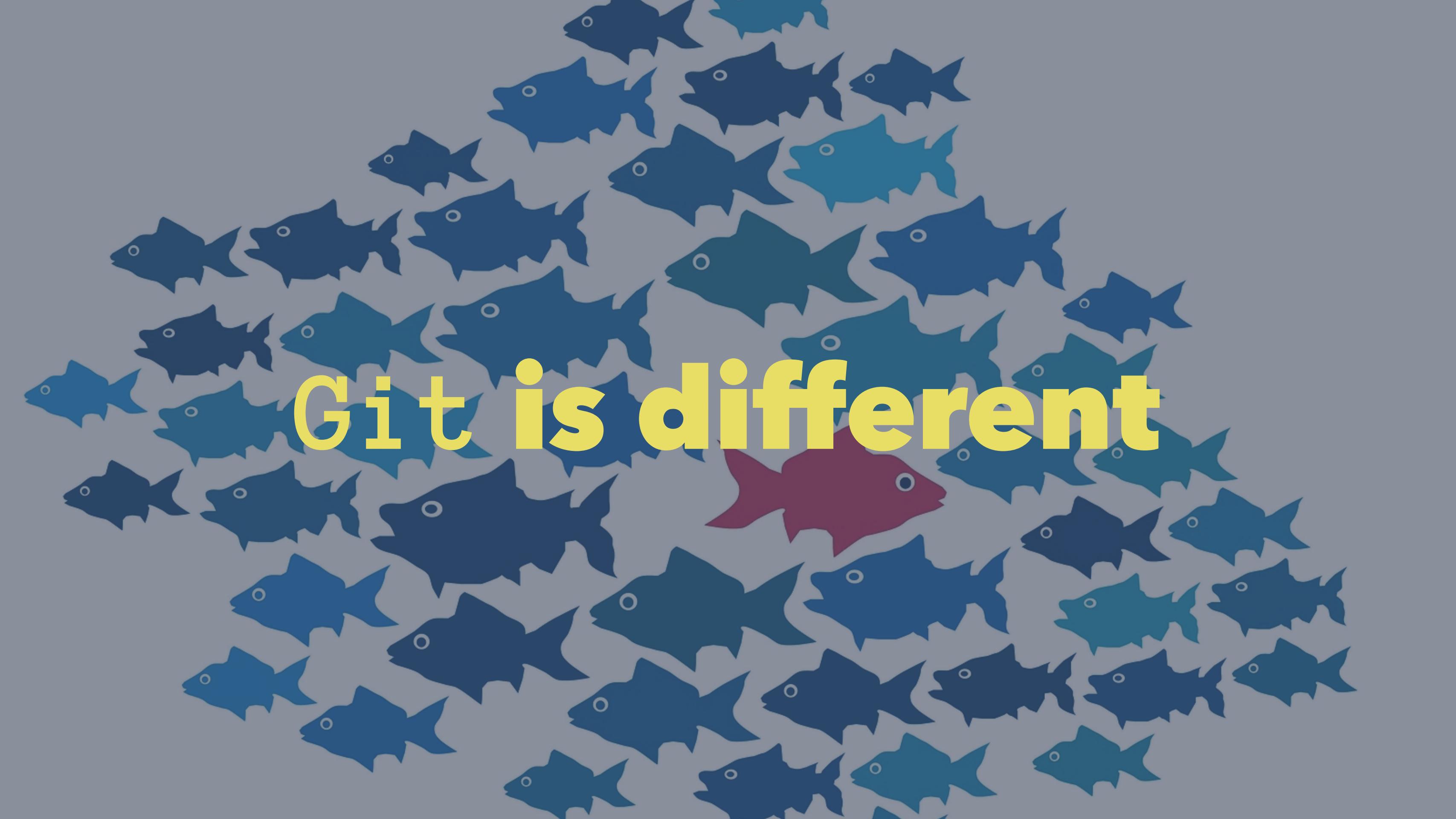
Staging  
Area

.git directory  
(Repository)



How to add changes?  
(Probably familiar with this)  
Follow up: How does Git track  
changes?

# **How does Git track changes?**



**Git is different**

Not only "Distributed"  
=> The way Git thinks about  
it's files

*The major difference [...] is  
the way Git thinks about  
its data.*

– **Scott Chacon & Ben Straub** ProGit

Checkins Over Time →



Deltas: What changed?

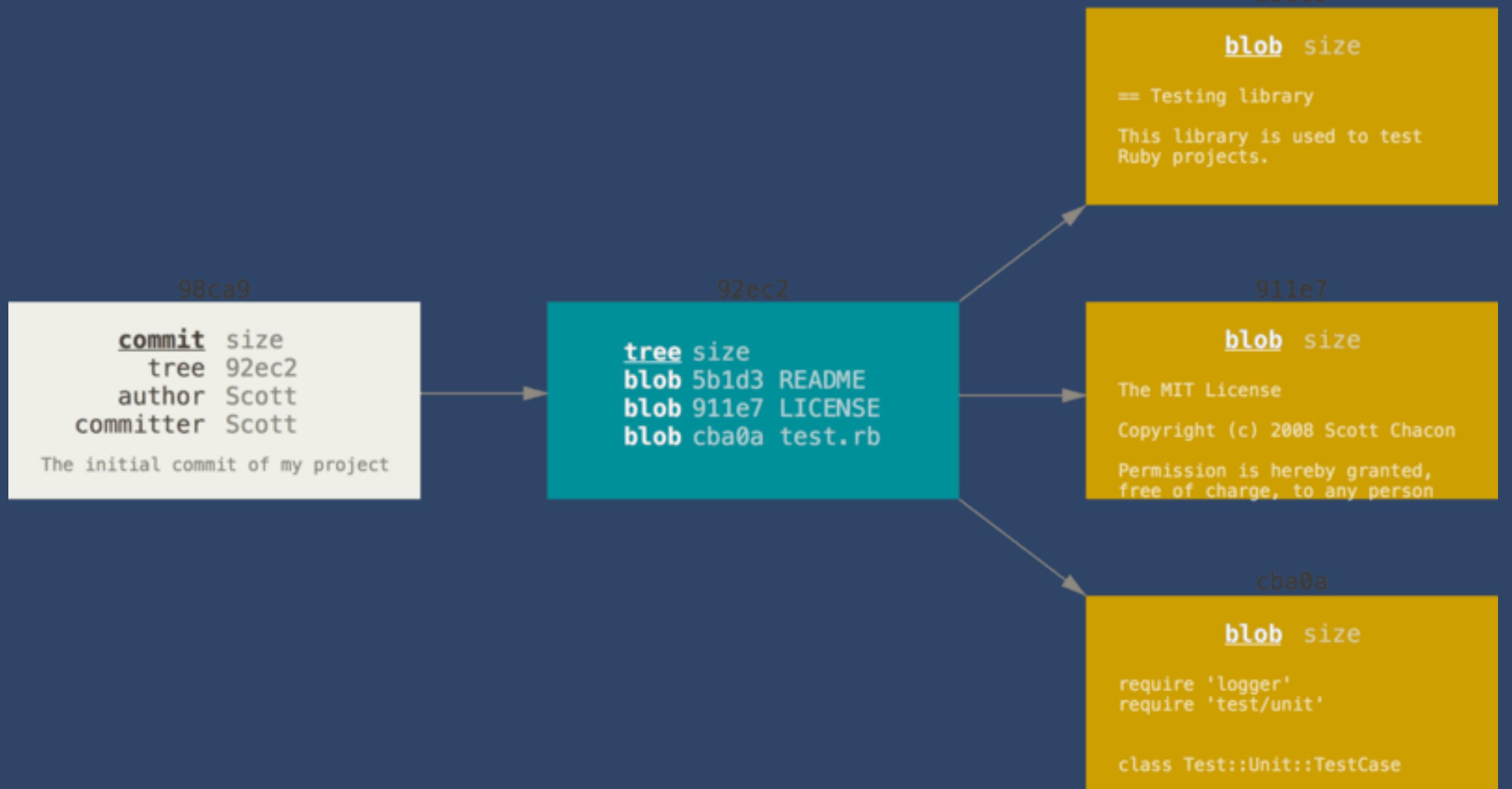
Snapshots: Current state

Q: Any advantages you can think of?

How does Git build this to a commit?

***What is a***

# **Commit**



Commit -> Tree -> Tree | BLOB  
Commit: Tree + Parents +  
Metadata

Git is closer to a filesystem then to  
a classic VCS

Next slide: How does Git identify  
objects?

# SHA1

Hash built over all properties of the object (Commit => Author, Committer etc.)

You might have heard to never change commits after pushing:  
Hash changes when properties change

# Try it out!

# Clone workshop repository

```
git clone  
git@github.gcxi.de:swolf/git-workshop.git
```

```
$ cd git-workshop  
  
$ echo 'Some random text' > my_file  
  
$ git hash-object -w my_file  
1a76b8a41993e2c667f5b191fb57abdab2102a8b  
  
$ git cat-file -t 1a76b8a41993e2c667f5b191fb57abdab2102a8b  
blob  
  
$ git cat-file -p 1a76b8a41993e2c667f5b191fb57abdab2102a8b  
Some random text
```

You can actually find the blob  
in `.git/objects/1a/`  
`76b8a41993e2c667f5b191fb57`  
`abdab2102a8b`

Git only uses the content; create  
an identical file and the hash will  
be the same

→ git log --all --decorate --graph --oneline

- \* f097585 (**HEAD → waldo-came-and-left, origin/waldo-came-and-left**) Add some more distracting stuff
- \* 0356fc5 Add some distracting stuff
- \* 8a28c0f Waldo left the building
- \* eeaf062 Add company for waldo
- \* c85f3cf Add some more distracting stuff
- \* d175476 Add waldo's name to the file

| \* f20c965 (**origin/master, master**) Merge branch 'develop' into 'master'

# How history is made

# How history is made

# git history at least



Each commit has a reference  
on it's parent => Single linked  
list  
Not really single though ...

# Honour thy parents

**Navigate through history**

Q: You want to remove the last commit, how?

**Λ**

***and***

**~<Ν>**

A composite image featuring a horse's head with a cat's face superimposed on it. The horse is wearing a bright orange hooded garment. It stands on a wooden railing of a ship, with the ocean and a compass rose visible in the background.

wat

Better have an example

```
$ git checkout ancestry
```

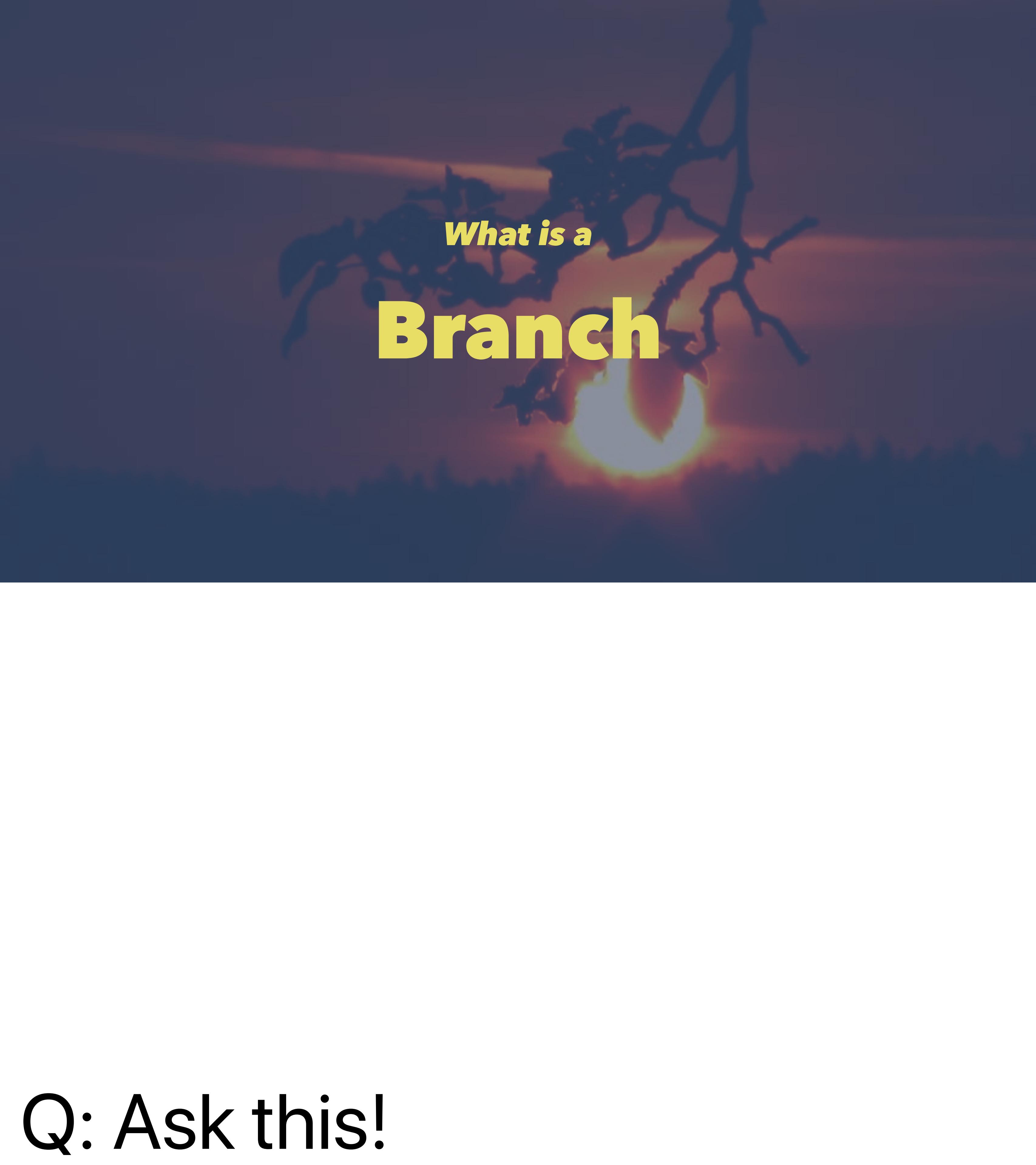
```
...
```

```
$ git log --oneline -1 HEAD  
822eca1 Add the number 5 to numbers
```

```
$ git log --oneline -1 HEAD^  
d4bd446 Add the number 4 to numbers
```

```
$ git log --oneline -1 HEAD^^  
1b94d52 Add the number 3 to numbers
```

```
$ git log --oneline -1 HEAD~2  
1b94d52 Add the number 3 to numbers
```



*What is a*  
**Branch**

**Q:** Ask this!

# Let's take a look

```
$ ls -l .git/refs/heads
total 32
-rw-r--r-- 1 swolf  staff  41 Jan 16 09:06 ancestry
-rw-r--r-- 1 swolf  staff  41 Jan 16 07:00 dev
-rw-r--r-- 1 swolf  staff  41 Jan 16 07:00 master
-rw-r--r-- 1 swolf  staff  41 Jan 16 07:02 waldo-came-and-left
```

All references in `.git/refs`:

- branches in `heads`
- remote branches in  
`remotes/<remote-name>`
- tags in `tags`

**Huh, master is a file ...**

**Q: What do you think is in  
master?**

# I wonder what's in there ...

```
$ cat .git/refs/heads/master  
f20c96538c6dca6fb37e631388814fe941afc2ae
```

# **It's just a hash!**

Let us inspect the object!  
We can use `git cat-file --batch`

```
$ cat .git/refs/heads/master | git cat-file --batch  
f20c96538c6dca6fb37e631388814fe941afc2ae commit 540  
tree 9ab6d12a4587fb4a02de1b5e745277c6c832cf5a  
parent c9866a994539e01938ceec6c75ada013ce456e9b  
parent 12e86b3370f931c423022a8c8d7d7fa7413e0bc4  
author Sascha Wolf <sascha.wolf@grandcentrix.net> 1516082444 +0100  
committer Sascha Wolf <sascha.wolf@grandcentrix.net> 1516082444 +0100  
gpgsig -----BEGIN PGP SIGNATURE-----
```

```
iHUEABEIAB0WIQTQyMji07ff76Vkk26j80vFo8w6AAUCW12VDAAKCRCj80vFo8w6  
AOqtAPoCPrGCN1QZKKGmlqZu43n824v2wviWbUujd9CBwuUURQD/dn2EJ2zHi+zQ  
qBKm1cfsi5vwTiYj3104Ub0Z1rJbw1M=  
=vi7g  
-----END PGP SIGNATURE-----
```

```
Merge branch 'dev'
```

# **What about HEAD?**

**Q:** What do you think is HEAD?

**Q:** Have you ever seen "detached HEAD"?

```
$ cat .git/HEAD  
ref: refs/heads/master  
  
$ git checkout --detach master  
...  
  
$ cat .git/HEAD  
f20c96538c6dca6fb37e631388814fe941afc2ae
```

1. Reference onto a branch
2. Reference onto a commit  
(detached HEAD)



*What is a*  
**Merge**

**Q:** How does a merge differ  
from a usual commit?

# What does the log tell us?

```
$ git log --decorate --graph --oneline
*   f20c965 (origin/master, master) Merge branch 'dev'
|\ \
| * 12e86b3 (origin/dev, dev) Add bar
| * 4043e69 Add some content to foo
|/
* c9866a9 Add foo
```

# **Remember the commit object?**

```
$ cat .git/refs/heads/master | git cat-file --batch  
f20c96538c6dca6fb37e631388814fe941afc2ae commit 540  
tree 9ab6d12a4587fb4a02de1b5e745277c6c832cf5a  
parent c9866a994539e01938ceec6c75ada013ce456e9b  
parent 12e86b3370f931c423022a8c8d7d7fa7413e0bc4  
author Sascha Wolf <sascha.wolf@grandcentrix.net> 1516082444 +0100  
committer Sascha Wolf <sascha.wolf@grandcentrix.net> 1516082444 +0100  
gpgsig -----BEGIN PGP SIGNATURE-----
```

```
iHUEABEIAB0WIQTQyMji07ff76Vkk26j80vFo8w6AAUCW12VDAAKCRCj80vFo8w6  
AOqtAPoCPrGCN1QZKKGmlqZu43n824v2wviWbUujd9CBwuUURQD/dn2EJ2zHi+zQ  
qBKm1cfsi5vwTiYj3104Ub0Z1rJbw1M=  
=vi7g  
-----END PGP SIGNATURE-----
```

Merge branch 'dev'

**Q: Do you notice something?  
=> Two parents**

# Tips & Tricks

# We're going to look at

- bisect
- --patch mode
- stash
- reflog
- rebase
- filter-branch

**One thing before we start ...**

Use aliases for interesting  
command+option  
combinations

# Aliase

```
git config --global alias.<your-alias> <command>
```

# In Action

```
$ git config --global alias.l 'log --decorate --graph --oneline'  
  
$ git l -3  
*   f20c965 (HEAD -> master, origin/master) Merge branch 'dev'  
|\ \\  
| * 12e86b3 (origin/dev, dev) Add bar  
| * 4043e69 Add some content to foo  
|/  
|/
```

--global to put it into your  
~/.gitconfig

A close-up photograph of a shotgun's barrel and receiver resting on a dark, ribbed surface, possibly a gun case. The shotgun has a dark finish and a visible serial number on the receiver.

git bisect

**a smart bug hunters weapon**

# git bisect

*Search for the first commit which introduces an issue*

```
$ git bisect start  
$ git bisect bad <known bad commit>  
$ git bisect good <known good commit>
```

## Shortcut

```
$ git bisect <known bad commit> <known good commit>
```

# Automate it!

`git bisect run <command>`

Exit codes are used

1: bad commit

0: good commit

# Let's try it out!

```
git bisect start waldo-came-and-left d175
```

Show the asciinema as demonstration at the end

# Stop bisecting

git bisect reset



---patch

add | checkout | reset

# Let's try it out!

```
$ git checkout patch-practice
```

```
...
```

```
$ git apply patch-practice.diff
```

```
...
```

```
$ git add --patch # Try to add only the above line!
```

Tip 1: ? for help

Tip 2: S for split



**Level up your --patch**  
**edit mode**

It's possible to edit things  
before adding it to the index!  
=> Show it on the example  
patch



git stash  
**quicksave for git**

Q: Who knows this already?

# git stash

- push create a stash from current changes
- pop apply and delete a stash
- drop delete a stash

Use `stash@{<n>}` to reference older stashes

# git reflog

## a history for your HEAD

# Try it out!

git reflog

# **Branches have reflogs too!**

`git reflog <some branch>`

# git rebase

**one command to rule them all**

Q: What rebase do?

feature/login-button

git rebase dev feature/login-button



# Let's try it out

```
$ git checkout rebase-practice
```

```
...
```

```
$ git rebase dev
```

```
???
```

Huh, seems like we got a conflict ...  
foo was removed but we want to keep it!  
Q: What can we do?



git rebase --interactive

# Let's fix this

We don't want to remove foo!

```
$ git rebase --abort
```

...

```
$ git rebase --interactive dev
```

???

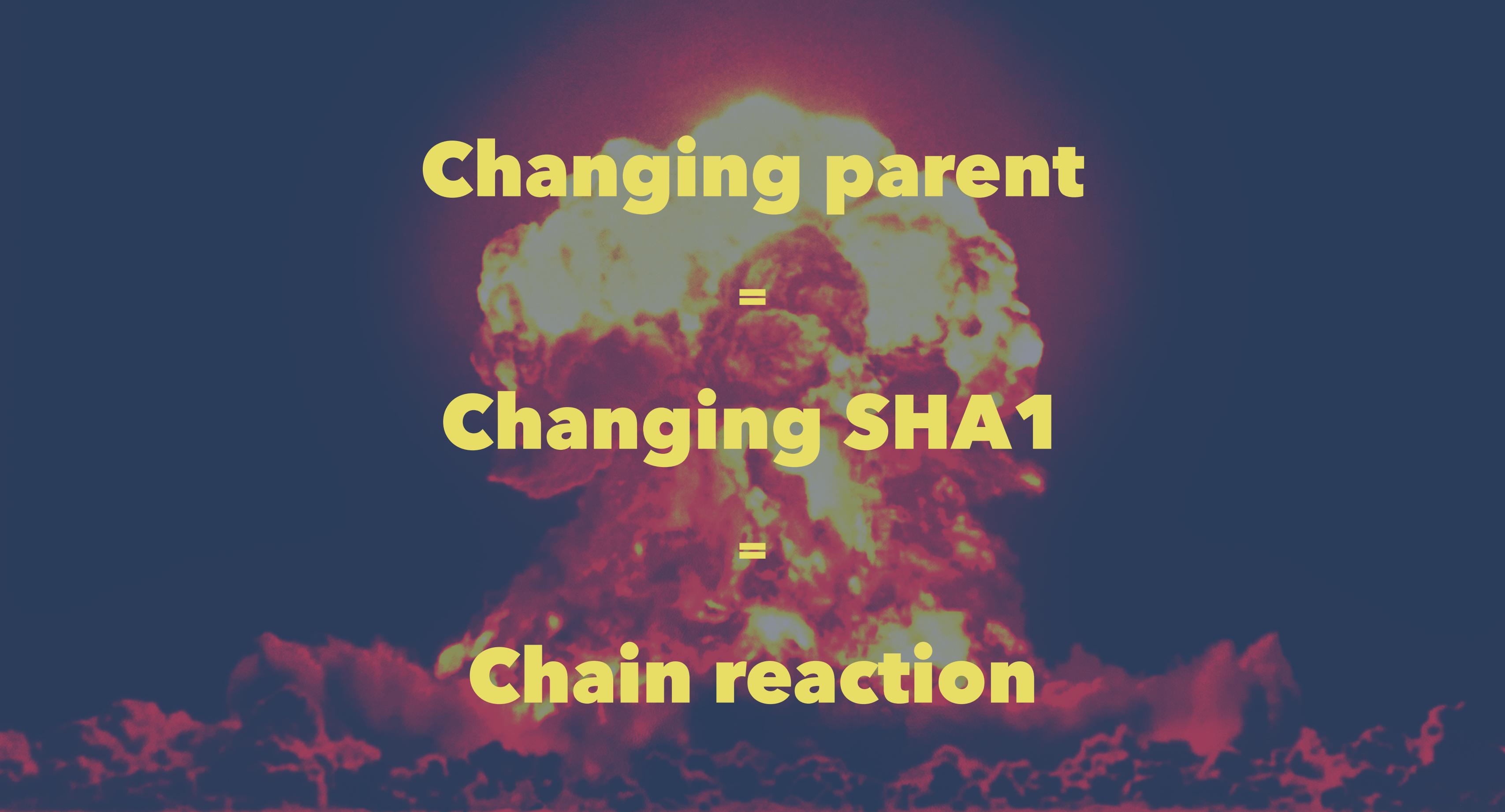
**Avoid rebasing pushed commits!**

Q: Do you know why?

# **rebase rewrites history**

**with great power comes great responsibility**

**Q: Why is that bad?**



**Changing parent**

=

**Changing SHA1**

=

**Chain reaction**



# MOAR POWER!

over history

git filter-branch

# REWRITE



Lot's of ways to rewrite history

# git filter-branch

Allows to rewrite

```
--env-filter <command>
--index-filter <command>
--msg-filter <command>
--parent-filter <command>
--subdirectory-filter <directory>
--tag-name-filter <command>
--tree-filter <command>
```

Need to remove a sensitive file  
from all commits?  
`tree-filter` to the rescue

# Honourable mentions

- cherry-pick: "copy" one+ commits onto the current branch
- clean: remove untracked files and folders
- commit --amend: include added changes in last commit
- grep: search pattern in tracked files
- worktree: checkout a reference into another worktree

# Best Practices<sup>2</sup>



---

<sup>2</sup> <http://sethrobertson.github.io/GitBestPractices>



# IN CASE OF FIRE



Git Commit



Git Push



Git Out

**Commit often  
Perfect later  
Publish once**

You can use interactive rebase  
for "Perfect later"

# **Use shortlived branches**

**or at least merge upstream work regularly**

**Do not rewrite  
published history!**

N: Otherwise your colleagues  
might do this



ORBO

# Commit Messages



# Commit Messages

<Topic|File>: <Short description> (50 characters sort limit)

<More detailed description> (bullet points are fine)

- problem (what is the issue with the current implementation)
- chosen approach with reasoning
- maybe rejected approaches

## Use imperativ

Add foo instead of Added foo

## N: Don't do this

## COMMENT

## DATE

O CREATED MAIN LOOP & TIMING CONTROL  
O ENABLED CONFIG FILE PARSING  
O MISC BUGFIXES  
O CODE ADDITIONS/EDITS  
O MORE CODE  
O HERE HAVE CODE  
O AAAAAAAA  
O ADKFJSLKDFJSDFKLJ  
O MY HANDS ARE TYPING WORDS  
O HAAAAAAAAAANDS

14 HOURS AGO  
9 HOURS AGO  
5 HOURS AGO  
4 HOURS AGO  
4 HOURS AGO  
4 HOURS AGO  
3 HOURS AGO  
3 HOURS AGO  
2 HOURS AGO  
2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT  
MESSAGES GET LESS AND LESS INFORMATIVE.

A group of approximately ten people of various ages and ethnicities are gathered outdoors in a park-like setting with trees in the background. They are all smiling and appear to be at a social event or picnic. Some are wearing casual summer clothing like t-shirts and shorts, while others are in more formal attire like a suit and tie.

Thank you