

AND JULIA EVANS
BY KATIE SYLOR-MILLER



RECIPES FOR GETTING OUT OF A GIT MESS

OH SHIT, GIT!
⑩ \$ % #

<https://ohshit.git.com>
LOVE THIS?

what's this?

If you find git confusing, don't worry! You're not alone. People who've been using it every day for years still make mistakes and aren't sure how to fix them. A lot of git commands are confusingly named (why do you create new branches with git checkout?) and there are 20 million different ways to do everything.



\$ man git reset

In the first and second form, copy entries from <tree-ish> to the index. In the third form, set the current branch head (HEAD) to <commit>, optionally modifying index and working tree to match. The <tree-ish>/<commit> defaults to HEAD in all forms.

↑
is this even English?

This zine explains some git fundamentals in plain English, and how to fix a lot of common git mistakes.



Katie
creator of
<https://ohshitgit.com>



I did something terribly wrong,
does git have a magic
time machine?

Yes! It's called git reflog and it logs every single thing you do with git so that you can always go back.

Suppose you ran these git commands:

- ① git checkout my-cool-branch
- ② git commit -am "add cool feature"
- ③ git rebase main

Here's what git reflog's output would look like. It shows the most recent actions first:

- | | |
|-------------|------------------------------------|
| ③ rebase: | 245fc8d HEAD@{2} rebase -i (start) |
| ② commit: | b623930 HEAD@{3} commit |
| ① checkout: | 01d7933 HEAD@{4} checkout |
- ⋮

If you really regret that rebase and want to go back, here's how:

git reset --hard b623930

git reset --hard HEAD@{3}

2 ways to refer
to that commit
before the rebase

Table of Contents

* git fundamentals *

- 1 I need to change the message on my last commit!
- 2 I committed but I need to make one small change!
- 3 I accidentally committed to the wrong branch!
- 4 I have a merge conflict!
- 5 A branch is a pointer to a commit.
- 6 HEAD is the commit you have checked out.
- 7 Every commit has a parent.
- 8 Mistakes you can't fix.
- 9 I need to change the message on my last commit!
- 10 I committed but I need to make one small change!
- 11-12 I accidentally committed to the wrong branch!
- 13 I tried to run a diff but nothing happened!
- 14 I have a merge conflict!
- 15 I committed a file that should be ignored!
- 16 I rebased and now I have 1,000 conflicts to fix!
- 17 I want to split my commit into 2 commits!
- 18 I want to undo something from 5 commits ago!
- 19 I did something terribly wrong, does git have a

19

I want to undo something from 5 commits ago!
OH SHIT!!

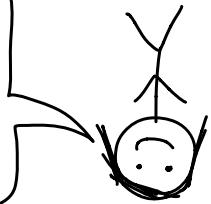
- 1 If you made a mistake but want to keep all of the commits since then, git revert is your friend!
- 2 git revert will create a reverse patch for the changes in a commit and add it as a new commit.
- 3 Find the commit SHA for the commit you want to undo.

git revert \$SHA_YOU_FOUND

② Run:

Now all of the changes you made in that commit are undone!

This is super useful if you push a bad commit to a shared repository and need to undo it!



- ③ Enter a commit message for the revert commit.

are undone!

◆ OH SHIT! mistakes & how to fix them ◆

- 4 A SHA is always the same code.
- 5 A branch is a pointer to a commit.
- 6 HEAD is the commit you have checked out.
- 7 Every commit has a parent.
- 8 Mistakes you can't fix.
- 9 I need to change the message on my last commit!
- 10 I committed but I need to make one small change!
- 11-12 I accidentally committed to the wrong branch!
- 13 I tried to run a diff but nothing happened!
- 14 I have a merge conflict!
- 15 I committed a file that should be ignored!
- 16 I rebased and now I have 1,000 conflicts to fix!
- 17 I want to split my commit into 2 commits!
- 18 I want to undo something from 5 commits ago!
- 19 I did something terribly wrong, does git have a

19



A SHA always refers to the same code

Let's start with some fundamentals! If you understand the basics about how git works, it's WAY easier to fix mistakes. So let's explain what a git commit is!

Every git commit has an id like 3f29abcd233fa, called a SHA ("Secure Hash Algorithm"). A SHA refers to both:

- the changes that were made in that commit see them with 'git show'
- a snapshot of the code after that commit was made

No matter how many weird things you do with git, checking out a SHA will always give you the exact same code. It's like saving your game so that you can go back if you die. You can check out a commit like this:

git checkout 3f29ab

↑
SHAs are long,
but you can just
use the first
6 characters

This makes it way easier to recover from mistakes!



I want to split my commit into 2 commits!

- ① Stash any uncommitted changes (so they don't get mixed up with the changes from the commit):

git stash

- ② Undo your most recent commit:

git reset HEAD^



safe: this points your branch at the parent commit but doesn't change any files

- ③ Use git add to pick and choose which files you want to commit and make your new commits!

- ④ Get your uncommitted changes back:

git stash pop



You can use git add -p if you want to commit some changes to a file but not others!

A branch is a pointer
to a commit

fix-fypd ↦ 775f06

awesome-feature ↦ 3bafea

main ↦ 2e9fab

A branch in git is a pointer to a commit SHA:

Here's some proof! In your favourite git repo, run
this command:

\$ cat .git/refs/heads/main

containing "2e9fab..."
this is a text file

- * git reset COMMIT_SHA will point the branch at COMMIT_SHA
 - * the remote branch
 - * git pull will point the branch at the same commit as the new commit
 - * git commit will point the branch at the new commit
- 3 main ways to change the commit a branch points to:
to figure out how to get your branch to point at the right
to fix your branches when they're broken: you just need
to understand what a branch is will make it MUCH easier

branches with many conflicting
commits, you can just merge!
alternatively, if you have 2

git rebase main
git merge-base
output of
goes here

git rebase -i \$SHA_YOU_FOUND

③ Squash all the commits in your branch together:

git merge-base main my-branch

② Find the commit where your branch diverged
from main:

git rebase --abort

① Escape the rebase of doom:

This can happen when you're rebasing many
commits at once.

I started rebasing and
now I have 100000000
conflicts to fix!



HEAD is the commit you have checked out

In git you always have some commit checked out. HEAD is a pointer to that commit and you'll see HEAD used a lot in this zine. Like a branch, HEAD is just a text file. Run `cat .git/HEAD` or `git status` to see the current HEAD.

Examples of how to use HEAD:

→ show the diff for the current commit:

```
git show HEAD
```

→ UNDO UNDO UNDO UNDO: reset branch to 16 commits ago

```
git reset --hard HEAD~16
```

HEAD~16 means
16 commits ago

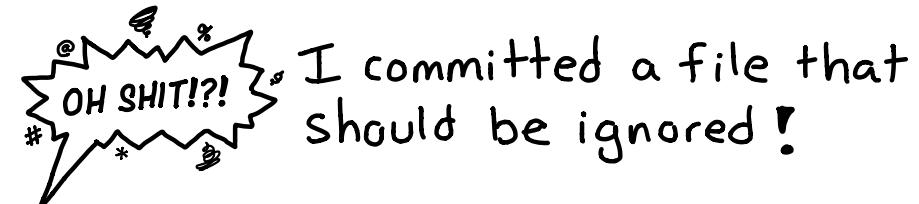
→ show what's changed since 6 commits ago:

```
git diff HEAD~6
```

→ squash a bunch of commits together

```
git rebase -i HEAD~8
```

this opens an editor,
use "fixup" to squash
commits together



Did you accidentally commit a 1.5GB file along with the files you actually wanted to commit? We've all done it.

① Remove the file from Git's index:

```
git rm --cached FILENAME
```

This is safe: it won't delete the file

② Amend your last commit:

```
git commit --amend
```

③ (optional) Edit your `.gitignore` so it doesn't happen again



now your coworkers won't be stuck
downloading a HUGE git commit!

git log shows you all the ancestors of the current commit, all the way back to the initial commit

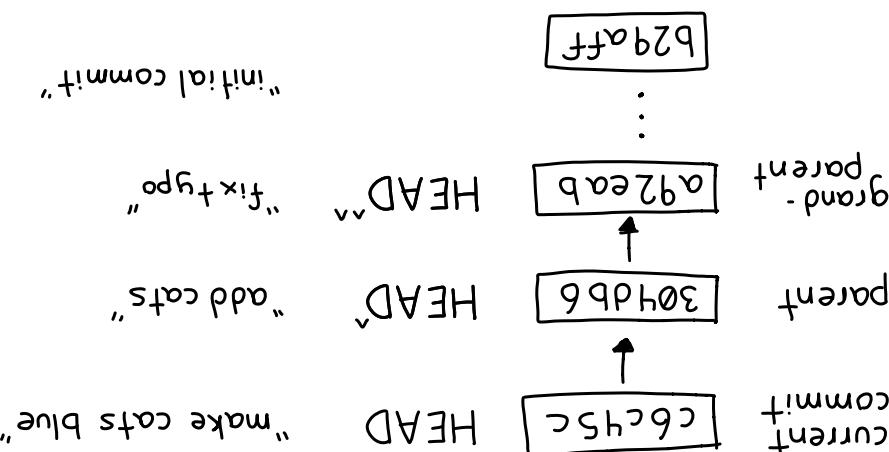
every commit has a parent
commits don't always have 1 parent. Merge commits

actually have 2 parents!



git checkout HEAD^

look at the code from the previous commit, you can run
git add the fixed files
git commit when you're done. → --fixup if you want to go
check out, and HEAD^ is its parent. So if you want to go



Every commit (except the first one) has a parent commit:
You can think of your git history as looking like this:

every commit has a parent

You can use a GUI to visually
resolve conflicts with git merge tool!
Mel (meldmerge.org) is a great choice!



① Edit the files to fix the conflict

```
<<<<< HEAD
if x == 0:
    return true
elif y == 6:
    return False
else:
    feature-branch
        code from main
    {
        code from main
    }
    >>>>> d34367
if x == 0:
    return True
elif y == 6:
    return False
else:
    feature-branch
        code from main
    {
        code from main
    }
    >>>>>
```

If that causes a merge conflict, you'll see
something like this in the files with conflicts:

git merge feature-branch.

Suppose you had main checked out and ran

I have a
merge conflict???



mistakes you can't fix

Most mistakes you make with git can be fixed. If you've ever committed your code, you can get it back. That's what the rest of this zine is about!

Here are the dangerous git commands: the ones that throw away uncommitted work.



git reset --hard COMMIT

- ① Throws away uncommitted changes
- ② Points current branch at COMMIT

Very useful, but be careful to commit first if you don't want to lose your changes!



git clean

Deletes files that aren't tracked by git.



git checkout BRANCH FILE or directory

Replaces FILE with the version from BRANCH.
Will overwrite uncommitted changes.



I tried to run a diff
but nothing happened?



did you know there are
3 ways to diff ?!

Suppose you've edited 2 files:

\$ git status

On branch main

Changes to be committed:

modified: staged.txt

Changes not staged for commit:

modified: unstaged.txt

staged changes
(added with 'git add')

unstaged
changes

Here are the 3 ways git can show you a diff for these changes:

→ git diff: unstaged changes

→ git diff --staged: staged changes

→ git diff HEAD: staged+unstaged changes

A couple more diff tricks:

→ git diff --stat gives you a summary of which files were changed & number of added/deleted lines

→ git diff --check checks for merge conflict markers & whitespace errors

I committed something to my main branch. I need to change the message on my last commit! OH SHIT!!

No problem! Just run:

`git commit --amend`

Then edit the commit message & save!

git commit --amend will replace the old commit with a new commit with a new SHA, so you can always go back to the old version if you really need to.

if you run `git commit`, but change your mind, you can always abort by deleting the commit message, saving, and quitting!



`git checkout -b` also checks out the branch create a new branch. The difference is `git branch` and `git checkout -b`, both



`git checkout my-new-branch`

④ Check out the new branch!

`git reset --hard HEAD~`

`git status`

careful!

③ Remove the unwanted commit from main:

`git branch my-new-branch`

② Create the new branch:

`git checkout main`

① Make sure you have main checked out:

I committed something to my main branch! main that should have been on a brand new branch!





I committed but I need to make one small change!



I accidentally committed to the wrong branch!

- ① Make your change
- ② Add your files with git add
- ③ Run:

```
git commit --amend --no-edit
```



this usually happens to me when I forgot to run tests/ linters before committing!

You can also add a new commit and use git rebase -i to squash them but this is about a million times faster.

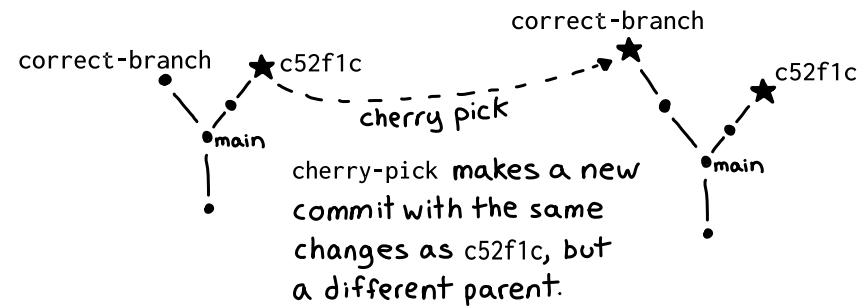
- ① Check out the correct branch

```
git checkout correct-branch
```

- ② Add the commit you wanted to it

```
git cherry-pick c52f1c
```

use 'git show wrong-branch' to find this



- ③ Delete the commit from the wrong branch

```
git checkout wrong-branch  
git reset --hard HEAD^
```



be careful when running 'git reset -hard'! I always run 'git status' first to check for uncommitted changes