Git tip of the week

Did you just mistakenly modify or delete a file in a git repo, and want to undo the changes?

If you haven't committed the file, then just do:

```
git restore <filename>
```

or

```
git checkout <filename>
```

And the file will be restored to its original state!

Git tip of the week

Did you just make some bad changes in a git repo, committed those bad changes to your local branch, and now want to undo the,?

If you haven't pushed the changes, then use:

```
git log --oneline
```

to find the SHA (first series of letters/numbers) of the commit just before your bad one, then do:

```
git reset <SHA>
```

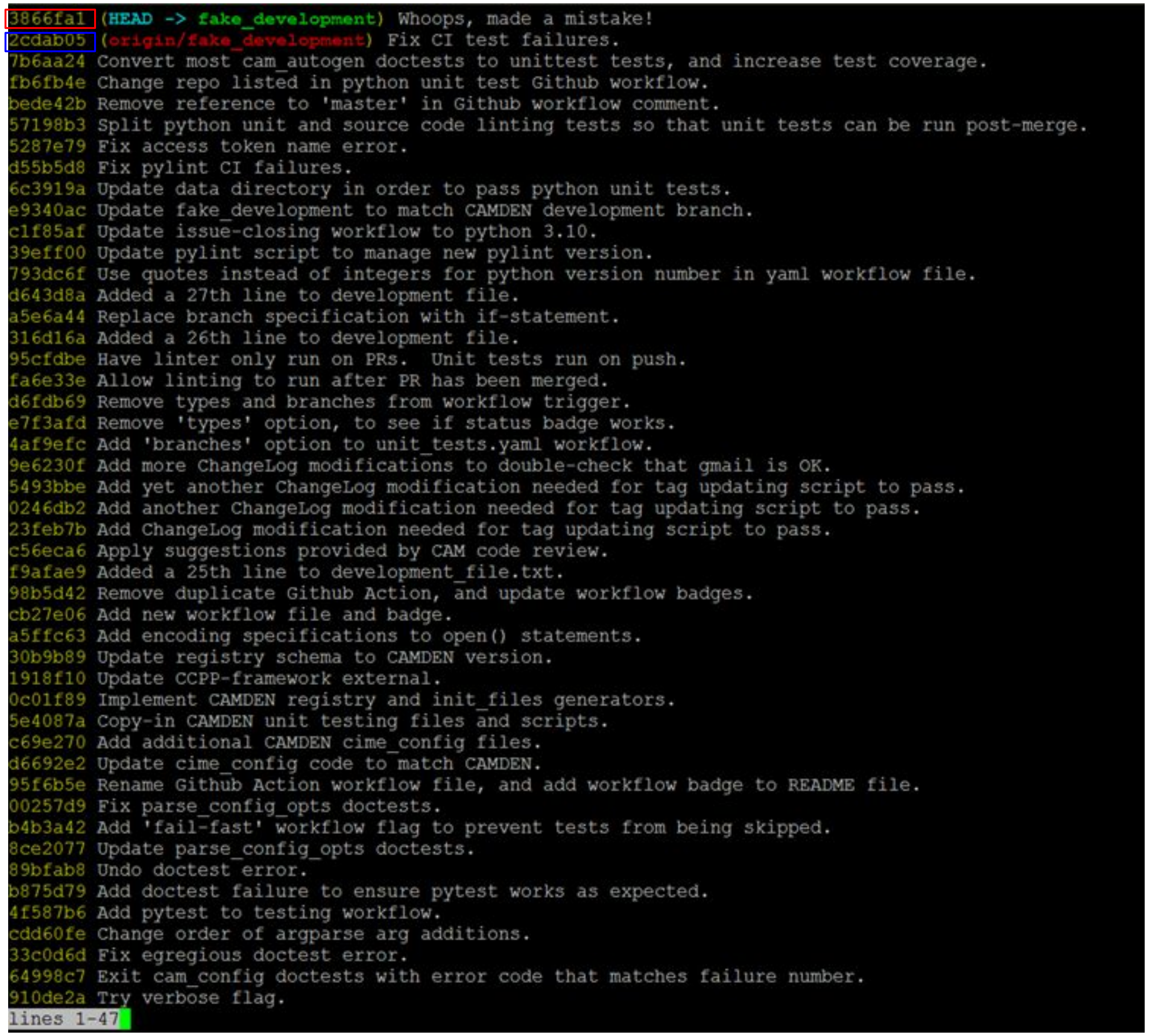
And the commit will be undone (but the changes will still be in the local files). If you want to completely blow away the changes as well, then do:

```
git reset --hard <SHA>
```

Git tip of the week

Bad commit

SHA to use in "git reset" call



```
3866fa1 (HEAD -> fake_development) Whoops, made a mistake!
2cdab05 (origin/fake_development) Fix CI test failures.
7b6aa24 Convert most cam_autogen doctests to unittest tests, and increase test coverage.
fb6fb4e Change repo listed in python unit test Github workflow.
bede42b Remove reference to 'master' in Github workflow comment.
57198b3 Split python unit and source code linting tests so that unit tests can be run post-merge.
5287e79 Fix access token name error.
d55b5d8 Fix pylint CI failures.
6c3919a Update data directory in order to pass python unit tests.
e9340ac Update fake_development to match CAMDEN development branch.
c1f85af Update issue-closing workflow to python 3.10.
39eff00 Update pylint script to manage new pylint version.
793dc6f Use quotes instead of integers for python version number in yaml workflow file.
d643d8a Added a 27th line to development file.
a5e6a44 Replace branch specification with if-statement.
316d16a Added a 26th line to development file.
95cfdbe Have linter only run on PRs.  Unit tests run on push.
fa6e33e Allow linting to run after PR has been merged.
d6fdb69 Remove types and branches from workflow trigger.
e7f3afd Remove 'types' option, to see if status badge works.
4af9efc Add 'branches' option to unit_tests.yaml workflow.
9e6230f Add more ChangeLog modifications to double-check that gmail is OK.
5493bbe Add yet another ChangeLog modification needed for tag updating script to pass.
0246db2 Add another ChangeLog modification needed for tag updating script to pass.
23feb7b Add ChangeLog modification needed for tag updating script to pass.
c56eca6 Apply suggestions provided by CAM code review.
f9afae9 Added a 25th line to development_file.txt.
98b5d42 Remove duplicate Github Action, and update workflow badges.
cb27e06 Add new workflow file and badge.
a5ffc63 Add encoding specifications to open() statements.
30b9b89 Update registry schema to CAMDEN version.
1918f10 Update CCPP-framework external.
0c01f89 Implement CAMDEN registry and init_files generators.
5e4087a Copy-in CAMDEN unit testing files and scripts.
c69e270 Add additional CAMDEN cime_config files.
d6692e2 Update cime_config code to match CAMDEN.
95f6b5e Rename Github Action workflow file, and add workflow badge to README file.
00257d9 Fix parse_config_opts doctests.
b4b3a42 Add 'fail-fast' workflow flag to prevent tests from being skipped.
8ce2077 Update parse_config_opts doctests.
89bfab8 Undo doctest error.
b875d79 Add doctest failure to ensure pytest works as expected.
4f587b6 Add pytest to testing workflow.
cdd60fe Change order of argparse arg additions.
33c0d6d Fix egregious doctest error.
64998c7 Exit cam_config doctests with error code that matches failure number.
910de2a Try verbose flag.
lines 1-47
```

Git tip of the week

Did you just mistakenly push a commit upstream (e.g. push something to the wrong repo), or want to fix a “bad” commit without completely erasing it?

You can undo the commit in your local code by doing:

```
git revert <SHA>
```

Where “<SHA> is the commit hash from git log. It will then display a commit message to accept or modify. Then you can push the revert upstream by doing:

```
git push
```

And the file will be restored to its original state both locally and upstream!

Git Merge vs Rebase

Git Merge	Git Rebase
Can update a branch with another branch/repo	Can update a branch with another branch/repo
Can be safely aborted if a conflict is found.	Can be safely aborted if a conflict is found.
Adds the updates as a regular commit (which can be reset/reverted if need be).	Re-writes the commit history to maintain a linear git history (harder to undo). Can be done "interactively" if you want to manually modify the history.
Changes are applied all at once.	Changes are applied one at a time.
Generally Safe to do on any branch/repo	Should usually only be done on personal branches/repos
Generally Safe to do anytime	Should never be done on a branch with an open PR

Git Merge vs Rebase

My personal workflow:

If I am on a branch on my fork that generally only I am working on, and I haven't opened a Pull Request (PR) from it yet, then I rebase.

For all other situations, I merge.

Git tip of the week

Did you change some file in a repo but cannot remember what you changed?

You can get a list of all modified files (that have not been committed) by doing:

```
git status
```

or alternatively:

```
git diff --name-only
```

Now let's say you want to know what was actually changed in the file. This can be done by typing:

```
git diff <name_of_file>
```

You can also just type "git diff" for all of the code changes across all files, but this can be difficult to read sometimes.

Git tip of the week

If you want to know what files were changed *after* committing, or want to just know what files are different between two different commits, then you can do:

```
git diff --name-only <SHA>
```

To get a list of all files, and:

```
git diff <SHA> <name_of_file>
```

To see how a particular file has changed between commits.

Git tip of the week

Do you ever do an operation in git and notice that it defaults to vim, and you would instead like to choose a different, slightly less cool editor instead?

Do you wish you could make your own git command aliases?

Would you prefer a different way to handle merge conflicts than the default?

Well all of this and more can be provided with:

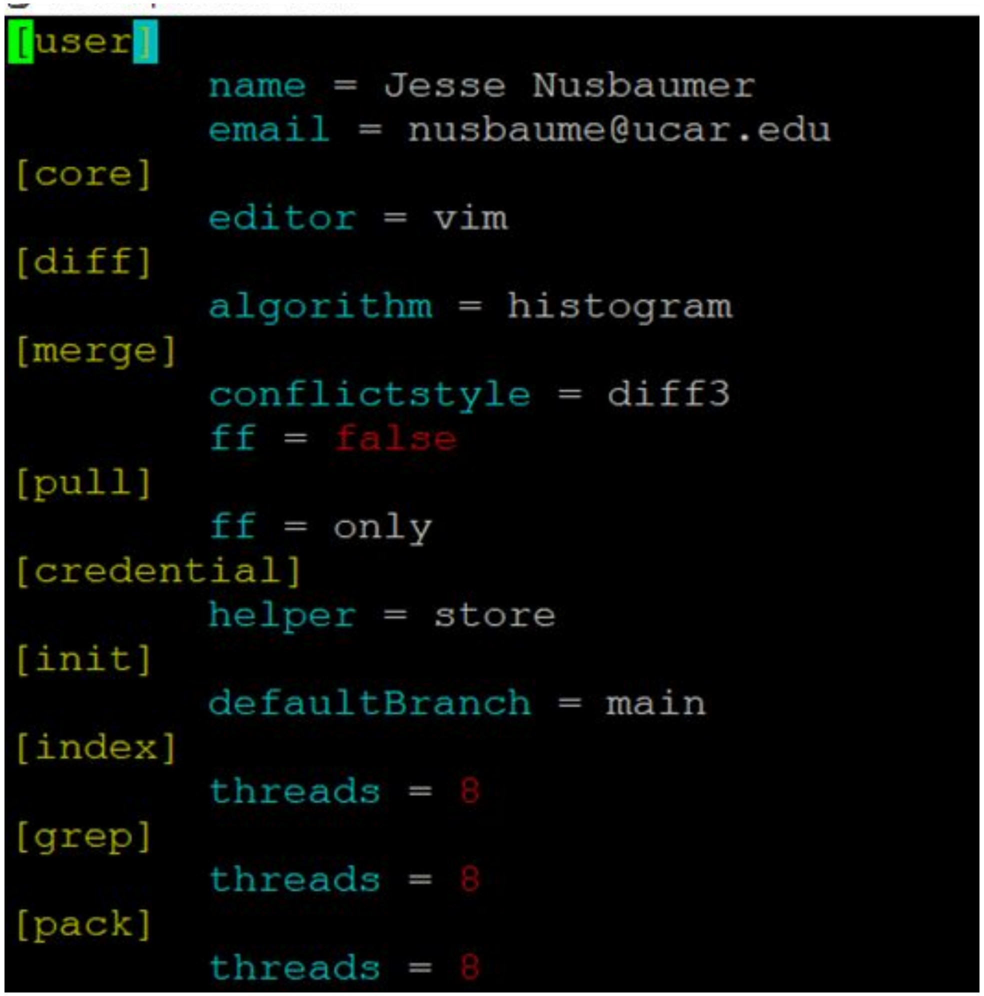
git config

Git tip of the week

The git config command creates either a ".gitconfig" file in your home directory, or a "config" file in a repo's ".git" directory, which contains a list of your specific git configurations for that computer system.

You can find a list of all possible git config options by running:

git help --config



```
[user]
        name = Jesse Nusbaumer
        email = nusbaume@ucar.edu
[core]
        editor = vim
[diff]
        algorithm = histogram
[merge]
        conflictstyle = diff3
        ff = false
[pull]
        ff = only
[credential]
        helper = store
[init]
        defaultBranch = main
[index]
        threads = 8
[grep]
        threads = 8
[pack]
        threads = 8
```

Git tip of the week

To change a git configuration for all your git repos on a given system, you can run the following on the command line:

```
git config --global  config_type.option X
```

For example, if I wanted to change the default editor to be emacs, I would do:

```
git config --global core.editor emacs
```

Or instead if I wanted to change my email that git uses (e.g. for commit messages), I would do:

```
git config --global user.email lulz420@hotmail.biz
```

Note not using "--global" will set the configuration for a specific git repo, instead of for all repos.

Git tip of the week

There are a myriad of possibilities for how one can configure git on a particular system. For new CAM developers we have a set of recommended config settings which can be found on Github here:

https://github.com/ESCOMP/CAM/wiki/git-and-GitHub#set-up-git-environment-on-new-machine

A list of popular git config options (found by Brian M.) can be found here: https://jvns.ca/blog/2024/02/16/popular-git-config-options/

Finally, Matt Dawson (SE in ACOM) provided the following website for git options (including command aliases) that he has used as well: https://lagrange.mechse.illinois.edu/git_quick_ref/

Git tip of the week

Do you ever have questions like:

What files did a commit change?

What commits have changed this file or directory?

Who was the last person to modify this line of code?

How can I look super cool in front of all of my peers and finally get the chance to hang out with the popular people?

Well git log and git blame can help you with most of those!

Git tip of the week

See what files were changed with each commit:

```
git log --stat
```

See what commits changed a specific file:

```
git log --stat --follow -- </path/to/file>
```

Or what commits modified files in a specific directory:

```
git log --stat --follow -- </path/to/direc>
```

What about examining individual lines in a file?

Git tip of the week

You can see info about the last commit, and the author of that commit, by doing the following:

```
git blame <file>
```

If you want to know more about the commit itself for a given line, then just take the provided hash and pass it to git log:

```
git log <hash>
```

Finally, let's say you want to know only the commits that added or removed a line in a file. This can be done, again, with git log:

```
git log -S <text string of interest> <file>
```

Git tip of the week

Let's say you have a friend, let's call him "Rich" (Github username "cricketfan"), who wants you to open a PR to his CAM branch "convect_or_die", which exists on his CAM fork. However, you already have your own CAM fork. What should you do?

You can add another person's branch to your own fork with the following commands (assuming you are in a local copy of your fork):

1. Add repo fork as a remote:
   ```
   git remote add richfork https://github.com/cricketfan/CAM.git
   ```
2. Get info from remote fork:
   ```
   git fetch richfork
   ```
3. checkout target branch on fork:
   ```
   git checkout richfork/convect_or_die
   ```
4. Create new local branch:
   ```
   git checkout -b convect_or_die
   ```
5. Push the branch to your own fork:
   ```
   git push -u origin convect_or_die
   ```

You now have your own copy of the branch which you can update and open PRs with!

Git tip of the week

Do you find with later versions of CAM or CESM that running git status takes forever?

This is because CESM now uses git submodules, and git status will go through the entire submodule tree, which can take…time…

To avoid this, try using the following command instead:

```
git status –ignore-submodules
```

If you use the command frequently then you can turn it into a git alias with the following command:

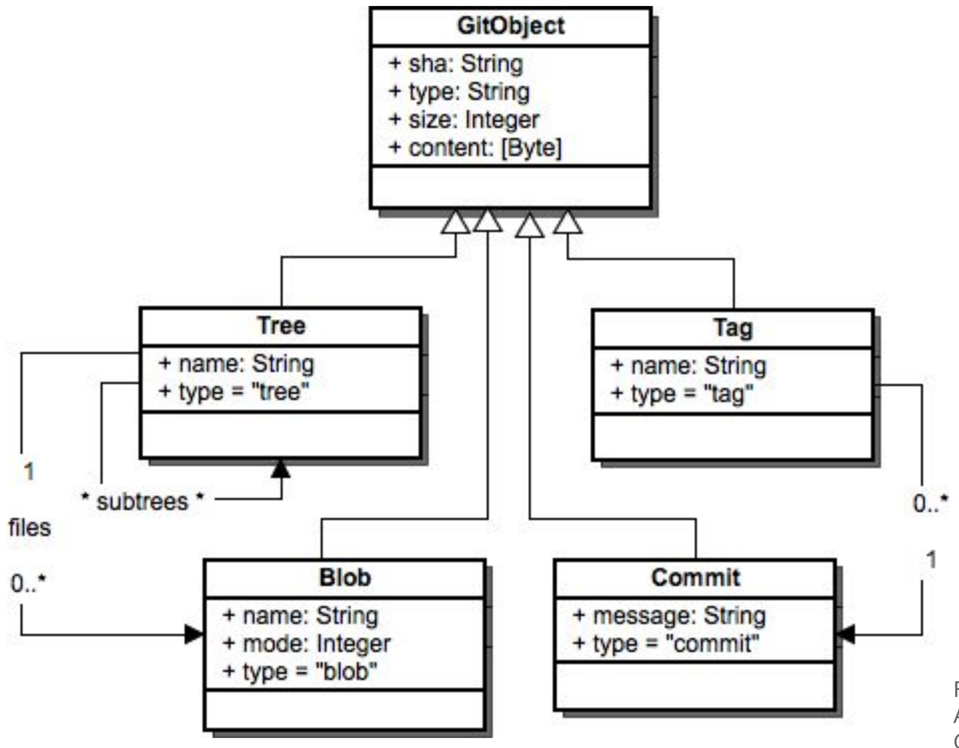
```
git config --global alias.stat 'status --ignore-submodules'
```

So now instead of "git status –ignore-submodules" just type

```
git stat
```

The git pit #1

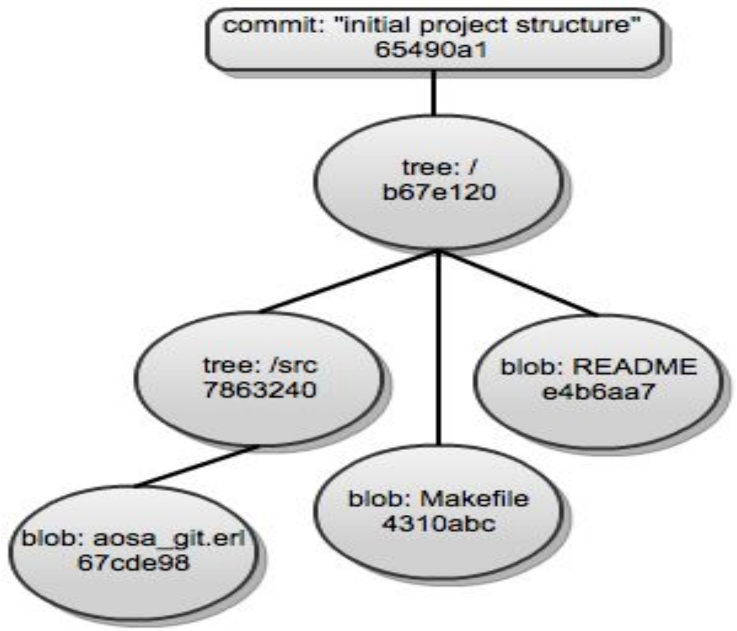
How does git actually store all of this info?
Everything is contained within four object classes



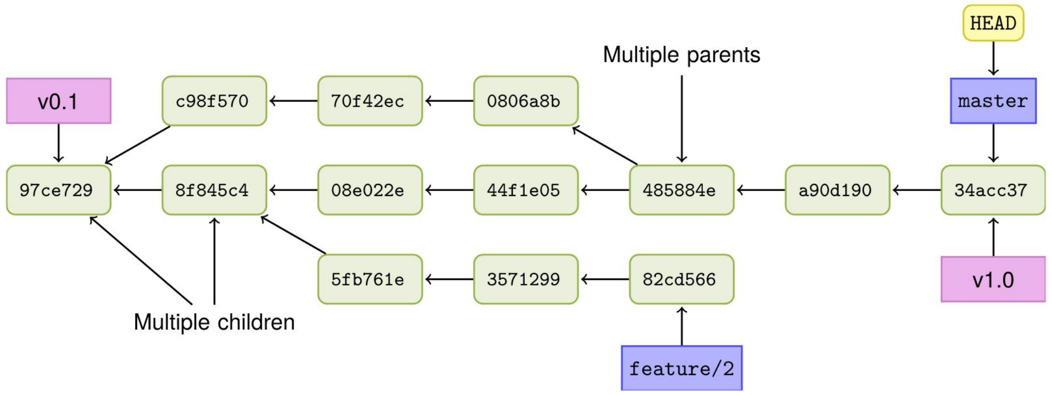
From The Architecture of Open Source Applications (Volume 2) - Git by Susan Potter

The git pit #2

How are those git object organized? In a Directed Acyclic Graph (DAG).



From The Architecture of Open Source Applications (Volume 2) - Git by Susan Potter



From https://devtutorial.io/git-is-a-directed-acyclic-graph-dag-p1182.html

Forbidden Knowledge

Beware the power of Git

Git tip of the week

Let’s say you just committed a change to a repo, but the commit message you wrote was not what you wanted, and you are trying to maintain the appearance of absolute perfection. How could you fix it?

To fix the most recent commit message, you just need to run the following:

```
git commit –amend
```

If you had already pushed the original commit back to a branch that currently only you are committing to, and your local branch is otherwise up-to-date, then you can force push the new commit message back to the repo:

```
git push -f
```

Git tip of the week

What if your git log messages show a never-ending series of cascading mistakes, and performance evaluations are due?

You can use git rebase to modify you commit history on your branch as far back as you want. The easiest way to do this is to run it interactively by selecting how far back from the head of the branch you want to go, and then typing

```
git rebase -i HEAD~n
```

where n is how many commits back from the head of the repo you want to go. Interactive rebases can allow you to change commit messages, combine commits together (squash), and even change the commit itself. When done you can then force push to your upstream branch (`git push -f`).

More info on both `git commit –amend` and `git rebase -i` can be found here:
https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History

ONLY EVER DO INTERACTIVE REBASE ON YOUR PERSONAL, NON-SHARED BRANCH. OTHERWISE IT CAN RESULT IN LOST COMMITS!!!!!!!