**Undoing Git**

1. **Accessing a past version**

If there are multiple commits, and we want to go back further than 1 commit, there is no short cut for that on Git. However, we can see all the previous commits made

**git log**

The changes are arranged in reversed chronological order (most recent first). Each commit will have a specific hash (commit <hash-is-here>), we can copy the commit hash and do:

**git reset <commit-hash>**

This goes back to the state of ALL FILES (if they are merged to create the changes).

1. **Checkout**

- To **see** the state for all files or 1 specific file at some time in the past, use

**git checkout <commit\_ID> <path/to/file/or/dir>**

and we can go back to the original version with **git checkout master**

**-** checkout is also used to discard the unstaged changes in the working repository

**git checkout <path/to/file/or/dir>**

1. **Reset**

If we want to unstage (not discarding)

**git reset <path/to/file/or/dir>** *(exclude file path if we want to undo every files)*

We can also use reset to uncommit local changes (not pushed)

**git reset HEAD~1** *(or <commit-hash> to specify when to reset to)*

what HEAD means is the pointer to the last commit (that we’ve just made). Therefore, if we want to go back to the previous commit, we use HEAD~1, which is 1 commit further. By doing this, we go back to when the changes are not staged and not committed. There are different ways in using git reset:

* ***git reset --soft HEAD~1****–*This command will remove the commit but would not unstage a file. Our changes still would be in the staging area.
* ***git reset --mixed HEAD~1*** *–*This is the default command that we have used in the above example which removes the commit as well as unstages the file and our changes are stored in the working directory.
* ***git reset --hard HEAD~1 –***This command removes the commit as well as the changes from your working directory. This command can also be called destructive command as we would not be able to get back the changes so be careful while using this command.

1. **Delete all new changes, go back to past version**

If we want to get rid of all the changes at some point. All the changes will not only be made unstaged, but also completely removed.

**git reset --hard <commit-hash>**

1. **Revert**

**git revert <commit\_ID>** *(or <path/to/file/or/dir>)*

This will **create a new commit** that inverts the changes in the commit specified 🡪 works even for commits that has been pushed.

Then, we still need to **git push** this new commit.

Meanwhile, git reset will only remove the commit locally, creating a conflict.

"Say we want to revert back to 0d1d7fc32. git log: a867b4af 25eee4ca 0766c053 0d1d7fc32"

|  |
| --- |
| # This will create three separate revert commits:  git revert a867b4af 25eee4ca 0766c053  # It also takes ranges. This will revert the last two commits:  git revert HEAD~2..HEAD  #Similarly, you can revert a range of commits using commit hashes (non inclusive of first hash):  git revert 0d1d7fc..a867b4a  # Reverting a merge commit  git revert -m 1 <merge\_commit\_sha>  # To get just one, you could use `rebase -i` to squash them afterwards  # Or, you could do it manually (be sure to do this at top level of the repo)  # get your index and work tree into the desired state, without changing HEAD:  git checkout 0d1d7fc32 .  # Then commit. Be sure and write a good message describing what you just did  git commit |

1. **Undo git init for the whole folder** 🡪 have to delete the whole .git file

**rm -rf .git**