**Undoing in Git - Reset, Checkout and Revert**

Git provides multiple methods for fixing up mistakes as you are developing. Selecting an appropriate method depends on whether or not you have committed the mistake, and if you have committed the mistake, whether you have shared the erroneous commit with anyone else.

**Fixing un-committed mistakes**

If you've messed up the working tree, but haven't yet committed your mistake, you can return the entire working tree to the last committed state with

$ git reset --hard HEAD

This will throw away any changes you may have added to the git index and as well as any outstanding changes you have in your working tree. In other words, it causes the results of "git diff" and "git diff --cached" to both be empty.

If you just want to restore just one file, say your hello.rb, use [git checkout](http://www.kernel.org/pub/software/scm/git/docs/git-checkout.html) instead

$ git checkout -- hello.rb

$ git checkout HEAD hello.rb

The first command restores hello.rb to the version in the index, so that "git diff hello.rb" returns no differences. The second command will restore hello.rb to the version in the HEAD revision, so that both "git diff hello.rb" and "git diff --cached hello.rb" return no differences.

**Fixing committed mistakes**

If you make a commit that you later wish you hadn't, there are two fundamentally different ways to fix the problem:

1. You can create a new commit that undoes whatever was done by the old commit. This is the correct thing if your mistake has already been made public.
2. You can go back and modify the old commit. You should never do this if you have already made the history public; git does not normally expect the "history" of a project to change, and cannot correctly perform repeated merges from a branch that has had its history changed.

**Fixing a mistake with a new commit**

Creating a new commit that reverts an earlier change is very easy; just pass the [git revert](http://www.kernel.org/pub/software/scm/git/docs/git-revert.html) command a reference to the bad commit; for example, to revert the most recent commit:

$ git revert HEAD

This will create a new commit which undoes the change in HEAD. You will be given a chance to edit the commit message for the new commit.

You can also revert an earlier change, for example, the next-to-last:

$ git revert HEAD^

In this case git will attempt to undo the old change while leaving intact any changes made since then. If more recent changes overlap with the changes to be reverted, then you will be asked to fix conflicts manually, just as in the case of resolving a merge.

**Fixing a mistake by modifying a commit**

If you have just committed something but realize you need to fix up that commit, recent versions of [git commit](http://www.kernel.org/pub/software/scm/git/docs/git-commit.html) support an **--amend** flag which instructs git to replace the HEAD commit with a new one, based on the current contents of the index. This gives you an opportunity to add files that you forgot to add or correct typos in a commit message, prior to pushing the change out for the world to see.

If you find a mistake in an older commit, but still one that you have not yet published to the world, you use [git rebase](http://www.kernel.org/pub/software/scm/git/docs/git-rebase.html) in interactive mode, with "git rebase -i" marking the change that requires correction with **edit**. This will allow you to amend the commit during the rebasing process.