Change a Commit Message that Hasn't Been Pushed Yet

```
git commit --amend -m "New message"
```

Add More Files and Changes to a Commit Before Pushing

```
# only do this BEFORE you've pushed the commits
git add -A

git commit --amend -m "My new message"
```

Remove Files from Staging Before Committing

```
git reset HEAD [filename]
```

Remove Changes from a Commit Before Pushing

```
# reset back to a specific commit
git reset HEAD~1

# or

git reset [HASH]
```

Remove Changes from a Commit Before Pushing

```
git reset --hard [HASH]

git reset --soft [HASH]

git reset --mixed [HASH]
```

Recover Local Changes from git reset --hard with git reflog

```
# To look up the commit hash
git reflog

git reset --hard [HASH]
```

Recover Local Changes from git reset --hard with git reflog

```
# To look up the commit hash
git reflog

git reset --hard [HASH]
```

Undo a Commit that has Already Been Pushed

```
# NOTE: Once a commit is pushed, do NOT use git reset
# make a "revert commit" to "undo" a specific commit

git revert [HASH-TO-UNDO]
```

Push a New Branch to GitHub that Doesn't Exist Remotely Yet

```
git checkout -b new-branch

git push

# Set the upstream of the local branch at the
same time

git push --set-upstream origin new-branch
```

Copy a Commit from One Branch to Another

```
git cherry-pick [HASH-TO-UNDO]
```

Move a Commit that was Committed on the Wrong Branch

```
# Get the commit we want
git cherry-pick [HASH-TO-UNDO]

# Remove the commit from the wrong  branch
git reset [HASH-TO-REMOVE]
```

Remove Files from Staging Before Committing

```
git reset HEAD [filename]
```

Use git stash to Save Local Changes While Pulling

```
# Save the local changes,
git stash

# Get remote changes
git pull

# To apply the stashed changed
git stash pop

# You will need to  fix the merge conflict
# Then drop the change from the stash
git stash drop stash@{0}
```

Explore Old Commits with a Detached HEAD, and then Recover

```
# checkout the hash of an old commit
git checkout [HASH]

# we'll be in a "detached HEAD" state
# Save the work by creating a new branch
git checkout -b my-new-branch
```

Fix a Pull Request that has a Merge Conflict

```
git checkout -b conflicts_branch

# Add 'Line4' and 'Line5'

git commit -am "add line4 and line5"
git push origin conflicts_branch

git checkout master

# Add 'Line6' and 'Line7'`
git commit -am "add line6 and line7"
git push origin master
```

Cleanup and Delete Branches After a Pull Request

```
# Locally confirm that remote is gone
git remote prune origin --dry-run
git remote prune origin

#clean up the feature branch
git branch -d feature-branch
```

Change the Commit Message of a Previous Commit with Interactive Rebase

```
git log --oneline

# start the interactive rebase

git rebase -i HEAD~3
# and then change pick to reword.
# We can now reword the commit message
```

git Ignore a File that has Already been Committed and Pushed

```
# We make a file and accidentally push it to github
# To remove it, add it to .gitignore file
# remove all of our files from our git cache
git rm -r --cached .

# add back all the files we want with
git add -A
```

Add a File to a Previous Commit with Interactive Rebase

```
git rebase -i HEAD~2

# during the interactive rebase, we can add
the file, and amend the commi
git commit --amend --no-edit

git rebase --continue
```

Fix Merge Conflicts While Changing Commits During an Interactive Rebase

```
# Enter interactive rebase
git rebase -i HEAD~2

# Then we can fix that merge conflict like normal
git rebase --continue
```

Squash Commits Before they are Pushed with Interactive Rebase

```
git rebase -i HEAD~3

# Make the changes in interactive rebase
# Make the commit message for that commit/save the message
# we'll be left with just a single commit
```

Completely Remove a File from Pushed git History

```
# prune history and garbage collect the remains
git reflog expire --expire=now --all && git gc
--prune=now --aggressive

# use git push to push that change to github
# and remove the .env file from all of the
history
```