

# Git Cheat Sheet

---

## A Typical Workflow

---

### Get updates from coworkers

```
git pull
```

### Make a new (local, private) branch and switch to it.

```
git branch new_branch  
git checkout  
  
//or, in one command  
git checkout -b new_branch
```

### See what has changed

```
git status
```

### Stage some of your changes for a nice, neat commit

```
git add file_one.txt  
git add file_two.txt  
  
//or, for a more interactive approach, to allow chunk-by-chunk seperation  
git add -p  
  
//then, commit them  
git commit -m "Succinct message about what happened"
```

### Save all your changes (locally)

```
git commit -a -m "Some message here"
```

## Share your recent commits (public branches only)

```
git push
```

## Publish a private branch

```
git push -u origin branch_name
```

# Other common commands

---

## Ignore changes to configuration files

```
git update-index --skip-worktree my_config_file.cfg
```

**Common use cases:** You have a configuration file that has a bunch of good default values, but developers will need to tweak some of these (e.g. build paths, library paths, OS-related flags). You want the file to be under version control, but any changes you make to be ignored

**Note:** things like `git reset` will **not** affect this file until

`git update-index --no-skip-worktree my_config_file.cfg` is executed. Other operations, like `git pull` will function as normally.

## Revert last commit (temporarily)

```
git revert HEAD
```

This creates a commit that undoes the changes you last made (additions become deletions and vice-versa). You can specify a range, or arbitrary

**Common use cases:** You want to split up the last commit into multiple or forgot to add files

## Undo last commit, preserve changes

```
git reset --soft HEAD~1
```

Technically, the `--soft` flag is on by default, but I can never remember if it is or not, so I always type it out.

**Common use cases:** You want to split up the last commit into multiple or forgot to add files

This completely erases the last commit you made, so do **not** use it if you have already pushed that commit.

## Permanently get rid of all changes since last commit

```
git reset --hard
```

**Common use cases:** You don't like the changes you've made and want to start over from the last commit

## Temporarily get rid of all changes since last commit

```
git stash
```

To undo:

```
git stash apply
```

**Common use cases:** You need to do something on a different branch, but can't automatically switch branches because of conflicts.

# Merging

---

To begin a merge, checkout the base branch (like master), and then make a temporary merge branch.

```
git checkout master          //we want to use master as our base
git checkout -b temp_merge

git merge branch_to_merge
```

If the merge succeeds automatically, you are done. Otherwise, you'll need to fix the conflicts (perhaps, using a merge tool) and then commit, like:

```
git mergetool
//fix conflicts

git commit          //no need to supply a message, git will generate one for you
```

## Merge strategies and strategy options

Sometimes you'll want to use the default recursive strategy with one of a few options like

```
git merge [branch_to_merge] -X ours           //If there are conflicts, resolve
git merge [branch_to_merge] -X theirs          //If there are conflicts, resolve
git merge [branch_to_merge] -X ignore-all-space //differences in whitespace are ig
git merge [branch_to_merge] -X patience        //can produce better diffs for mer

git merge [branch_to_merge] -s ours            //Ignore all changes in branch_to_
```

## Dealing with Forks

If you have forked a repository and still want to be able to get updates from that repository and merge them in with your own changes, you'll need add a link to the original repository. We call this adding remote repository (or remote, for short) that is, by convention, called **upstream**.


Adding the remote (only need to do this once):


```
git remote add upstream https://github.com/[original_username]/[repository_name].git
```


To get any changes the original owner has made to their repository, we need to first **fetch** them, then merge them.

```
git fetch upstream           //gets changes from original repo, but does **not** integ
git merge upstream/master    //Merges in the upstream changes
```


If you want to send your changes to the original repository, you'll need to publish a branch with your changes and then make a **pull request** at GitHub.com. On the website, change to the branch you want to merge and then, press the Compare & Review button:

 **89** commits

 **4** branches



branch: **test-branch** ▾

**testing** / 

**Compare & review**

This branch is **2 commits ahead and 2 commits behind master**