Rebasing

coursera.org/learn/git-distributed-development/supplement/ibg9w/rebasing

Suppose you establish a development branch at some point in time and the main branch that you began from continues to evolve at the same time you are making changes to your development branch.

Eventually, you intend to merge your work with the parallel development branches, but suppose your work is not quite ready for prime time, but you want to get the benefit of the other work going on in the parallel branch.

There are two basic methods that may seem similar, but are actually quite different; merging and rebasing. Merging simply means bringing in the other (probably mainline) evolving repository from time to time. Any conflicts will need to be resolved as usual.

Rebasing is quite different. When you follow this procedure, all your changes since your initial branch off the other line of development are reverted, the branch is brought up to date to its present state, and then your changes are reworked so they fit on the current state.

To give a concrete common example, suppose you are working on a new feature for the Linux kernel, and you began working when the kernel release status was version 4.16 . While you are happily coding away, the mainline kernel continues to evolve and version 4.17 is released. If you do a rebase, your changes, your patch set, are rewritten to apply to the 4.17 code rather than the 4.16 code, a hopefully simpler situation to deal with in the future. In fact, you may rebase further as time goes on.

The steps to a rebase are straightforward. Suppose at some point in the past you did:

```
$ git checkout -b devel origin
```

where **origin** is a remote tracking branch as we will discuss later (you could also use a local branch of course, but that would probably be less realistic). Then, you go about and make a series of changes to project files and make commits in your **devel** branch. Meanwhile, the **origin** branch continues to evolve.

The rebase operation begins with:

```
$ git checkout devel
$ git rebase main devel
```

When you do this, each commit done since the **original** branch point is removed, but is saved in **.git/rebase-apply**. The **devel** branch is then moved forward to the latest version of origin, and then each of the changes is applied to the updated branch.

Obviously, conflicts may emerge and if any are found you will be asked to resolve them as you did when you were merging. As you fix the conflicts, you will have to do **git add** to update the index, and when you are done fixing conflicts, instead of doing a commit, you do:

```
$ git rebase --continue
```

If things get disastrous somewhere along the way, you can revert back to where you were before the attempted rebase with:

```
$ git rebase --abort
```

There are potential problems associated with rebasing, some of which arise from its Orwellian nature. When you do a rebase, you change the history of commits, because the changes are temporarily removed and then all put back in. This leads to at least several problems:

- Presumably, you have been testing your work as you have been progressing it, and the code branch you were testing against came from the earlier branch point. There is no guarantee that just because things worked before when you tested they still will. Indeed, problems that arise may be very subtle.
- You may have done your work in a series of small commits, which is a good practice when trying to locate where problems may have been introduced, for instance when using bisection. But now you have collapsed matters into a larger commit.
- If anyone else has been using your work, has been pulling changes from your tree, you have just pulled the rug out from under their feet. In this case, many developers consider rebasing to be a terrible sin.

If you are doing merging rather than rebasing, other problems can arise, especially if you do it often or not at major stable development points. Once again, history can be confusing in your project. Whatever strategy you choose, think things through and only do merging or rebasing at good, well-defined stages of development to minimize problems.

Changes induced by **git rebase**:

Command	Source Files	Index	Commit Chain	References
git rebase	Unchanged	Unchanged	Parent branch commit moved to a different commit	Unchanged