Retrieving and Comparing Versions

Contents

Retrieving Past Versions

To cancel your latest commit (revert to the previous version), run the following command:

```
git revert HEAD
```

This command creates a new commit that reverts the changes made in the last version. If you want to retrieve a version from weeks or months ago, start by using `git log` to find the SHA of the version you want to retrieve. To reset your entire project to this version run the following commands:

```
git checkout SHA_of_the_version
```

If you want the old version of a single file and not the previous version of the entire project, you can do so by using the following command:

```
git checkout SHA_of_the_version -- your_file_name
```

Good Practice

Commits should be 'atomic', meaning that **they should do one simple thing and they should do it completely**. For example, an 'atomic' commit could be adding a new function or renaming a variable. If a lot of different changes to your project are all committed together, it can be hard to troubleshoot if any error appears in that version. Furthermore, undoing the whole commit may throw away valid and useful work.

It is good practice to **specify the files to be committed**, that is, adding files to the staging area by name (`git add your_file_name`) rather than adding everything (`git add .`). This prevents you from unintentionally bundling different changes together. For example, if you have made a change to file A while primarily working on file B, you may have forgotten this when you go to commit. With `git add .`, file A would be brought along for the ride. If there are several *unrelated* changes that should not be added together in a *single* file, `git add -p your_file_name` will let you interactively choose which changes to add. That said, **you do not necessarily need to do per-file commits** when working on multiple files, but for one single problem. For example, if we add a figure to this chapter here, choosing one to catch the attention of someone skimming through:

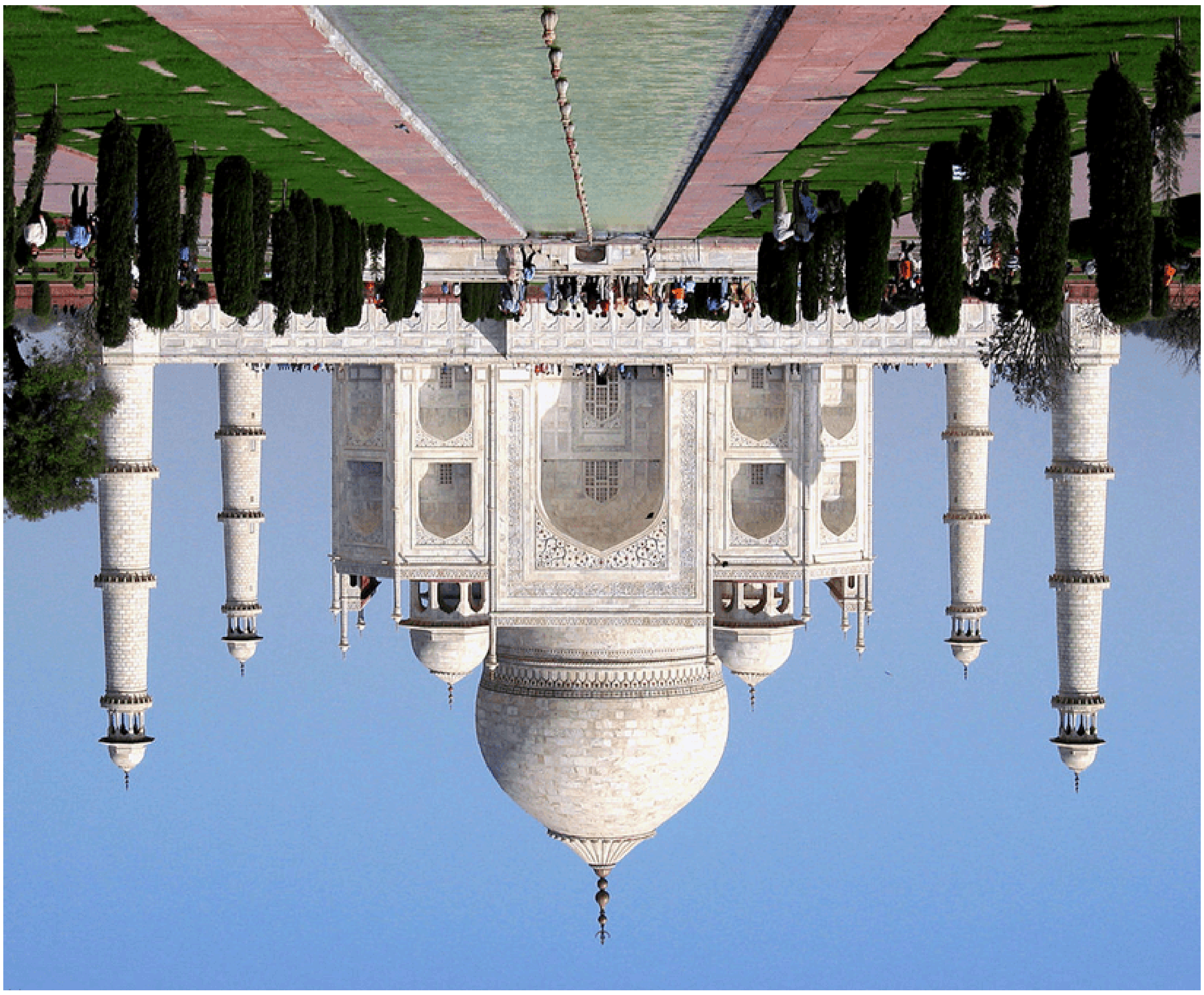

Fig. 31 Flipped Taj Mahal

two files are changed:

1. First, the figure file is added in the project repository.
2. Then, a line is added in this file that references the figure, so it is displayed.

So two files are affected, but “Add figure to version control chapter” is a single, *atomic* unit of work, so only one commit is necessary.

Finally, do not commit anything that is regenerated from other files committed in a version (unless it is something that would take hours to regenerate). Generated files, such as scripts, clutter up your repository and may contain features such as timestamps that can cause annoying file conflicts (see Merging Branches in Git). You can instruct Git to ignore certain files by creating a file called `.gitignore` and including names of the file that you do not need to store in your Git repository. For example, configuration files that might change from environment to environment should be ignored.

Comparing Versions

At some point, you will likely need/want to compare versions of a project, for example, to see what version was used to generate a particular result.

To address this issue, use the `git diff` function, that takes two input data sets and outputs the changes between them.

`git diff` is a multi-use function that runs on Git data sources such as commits, branches, files and more. By default, `git diff` will show you any uncommitted changes since the last commit. If you want to compare two specific things the syntax is:

```
git diff thing_a thing_b
```

For example, if you want to compare how a file has changed between two commits, use `git log` to get the SHAs of those commits and run:

```
git diff SHA_a:your_file_name SHA_b:your_file_name
```

Or if you wanted to compare two branches, it would be:

```
git diff branch_name other_branch_name
```

Good practice

With a little familiarity, `git diff` becomes an extremely powerful tool you can use to track what files have changed and exactly what those changes are. This is extremely valuable for unpicking bugs and comparing work done by different people. Be careful to **understand what exactly is being compared** and, where possible, **only compare the relevant files** for what you are interested in to avoid large amounts of extraneous information.