If you program long enough, there will come a time when you accidentally lose code that is important. Most frequently, this happens when you have an idea for how to improve your code that turns out not to work. You spend a lot of time reorganizing and rewriting the code only to find out that the final version is worse than what you started with. At that point, you want to go back to the original version. But, it's too late, as that code is now gone forever. Maybe you can rewrite it as it was, maybe it was too complicated and you can not. Either way, you have now wasted a lot of time. What can you do to prevent this? This is one of the many situations that can be helped by the judicious use of version control.

What is version control?

A simple way to understand version control is to think about keeping many different *versions* of your project. At regular intervals, perhaps whenever you get a new feature/function written, you take a snapshot of your project and save it away for later. You never edit these old snapshots. Rather, they are available for you to look at whenever you need to.

Return to our initial example. Before you try to improve your code, what if you save the current version? Then, when you later find out that you have taken the wrong approach, you can retrieve that old version and keep working from there. You have only lost the time exploring your improvement, not recreating your old code that worked better!

You can imagine doing this on your own by simply saving files away routinely. However, this is a recipe for failure. You will inevitably overwrite old files and not be able to find what you are looking for. Murphy's law says that it's very likely you will accidentally overwrite exactly the version you need in the future.

Version Control Systems

This is not a unique problem. Every programmer faces it. Therefore, it makes sense that people have developed tools to help. Version control systems give you a systematic way of saving the history of your project, simplify going back through your changes, and help you to minimize mistakes in working with versions.

There are many, many version control systems out there. Different people will tell you that different systems are "the best" and that you must use a particular system. This is largely false. The different systems basically have the same underlying models, but differ

in their approach to maintaining the version history. Currently, distributed version control systems, such as git, are very popular. GitHub is a popular version control hosting site that only supports git, further reinforcing the popularity of git.

However, it is more important that you *use* a version control system than *which* version control system you use. Github offers a wide range of conveniences, so is a natural choice for many people. However, there are also good reasons to use other version control systems, such as subversion. Rather than take a position on which system is better, we will simply encourage you to learn about version control system and use whatever system makes the most sense to you. If you are unsure, trying things out on Github is a perfectly reasonable first step. **However**, note that using git successfully can become extremely complicated. Git allows a wide range of workflows, so it is possible to find your project in an unintended state if you are not disciplined in your use of git. There are many resources you can find on the web, each advocating different ways of using git. Start with something simple that you can understand and build up from there.

Basic Concepts

There are a few basic concepts that transcend any particular version control system. Let's define some terms:

- Repository: The *repository* stores the entire history of your project. Ideally the repository is located on a server somewhere, so that you do not lose your work even if your computer is lost or breaks.
- Revision (or version): A *revision* is a particular historical (or current) version of your program.
- Working copy: A working copy is a local copy of the files from the repository that are stored on your computer. You make all edits as you develop your program on your working copy.
- Checkout: A *checkout* is an operation that creates a working copy from the repository. You can checkout the latest revision or a past revision.
- Commit: A *commit* is an operation that creates a new revision from all (or some) of the files in your working copy.

So, the general flow in using version control is as follows:

1. You create a new repository on a server somewhere (such as github.com).
2. You checkout a working copy of the repository on your computer.
3. You edit the files in the working copy (or add new ones).
4. You regularly commit your changes to create new revisions at meaningful times in the development of your program. You write informative commit messages that are attached to each revision so that you can easily understand what those revisions contain.

Keep in mind that these are abstract steps. A particular version control system, such as git, may require you to execute multiple commands to accomplish any particular step. Or it may allow additional steps.

Git 101

Git is currently one of the most popular version control systems. It is also very complex. You should start with the basics and work from there. We recommend you search for "git tutorial for beginners" or the like to find resources online. One of the easiest ways to start with git is to use an online service to create a repository. Go to that service, create an account, and create a new repository. You should install git on your computer.

Once you have git installed, you can use the following commands (either at a command prompt or through a graphical program that you can install):

git clone <repository>

Here "<repository>" is the address of the repository on the server. Whatever online service you choose should clearly tell you what to use as the repository in this command.

git add <file(s)>

Here <file(s)> is the name of whatever file or files you have changed that you want to add to the next revision. Note that these files can be new files that are not already in the repository or they can be modified files.

git commit

This will take all of the added files and create a new revision. Note that this revision only exists on your local computer at this point, it does not yet exist on the repository on the server.

git push

This takes all of the commits on your local computer and *pushes* them up to the repository on the server.

If you have the repository cloned on multiple computers, then you can do *git pull* to retrieve all of the commits that were pushed to the server from another clone. Note that if you make commits on multiple computers, you may get into a situation where you have conflicts (if you edit the same file in multiple clones without pushing/pulling between the edits). Understanding, handling, and resolving conflicts is beyond the scope of this simple introduction.

Working With Others

Another thing that version control systems do well is enabling you to work well with others. If multiple people are working on the same project, imagine the chaos that would ensue if two people make large numbers of edits to the same file. How would you figure out how to merge all of those edits? All version control systems have components that help you with such tasks.

When working with other people, you would each clone the same git repository and work on your respective working copies. You would make commits as normal and push your changes back to the repository on the server. You can get other people's work simply by pulling their work from the server. Again, though, you could end up with conflicts where multiple people made different edits to the same code.

When working in groups, there is no substitute for good version control habits. Otherwise, it quickly becomes very difficult to successfully share code and develop software.

Where to go from here?

This was meant to be a very *brief* introduction to version control concepts, not a complete tutorial or instruction manual. When you are ready to try to use version control, make sure that you find some tutorials that make sense to you. You have also learned how valuable documentation can be. Don't shy away from reading the documentation!

Mark as completed