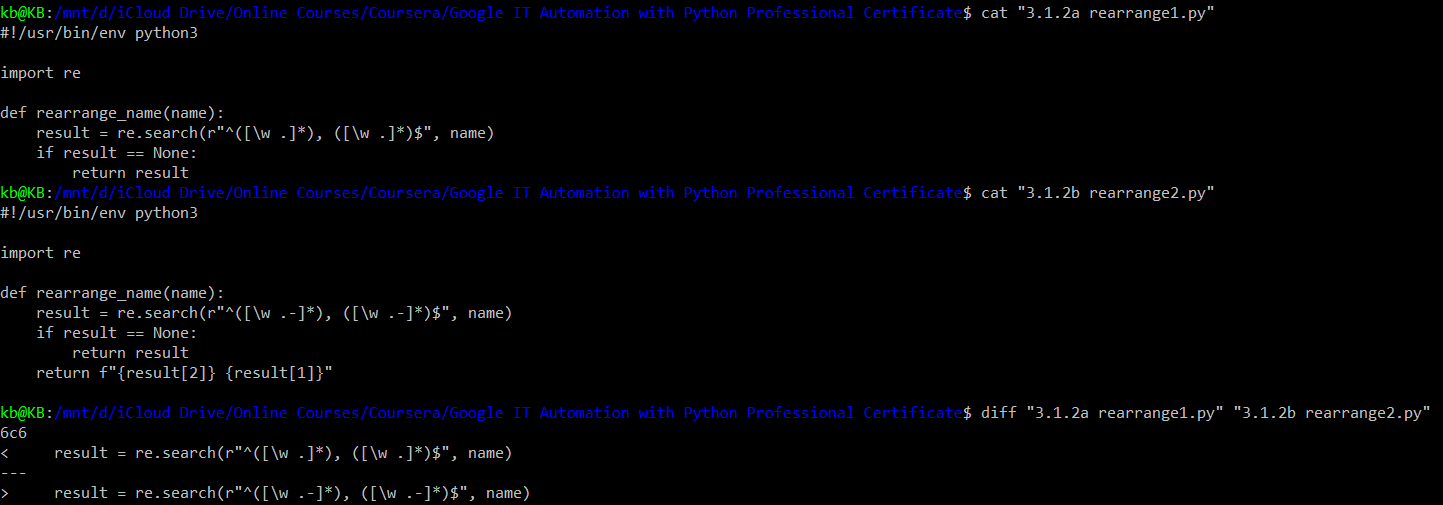
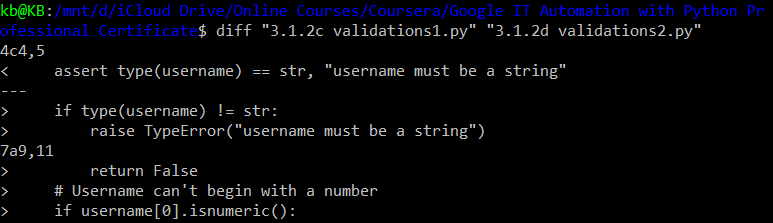
Before Version Control

**Diffing Files**

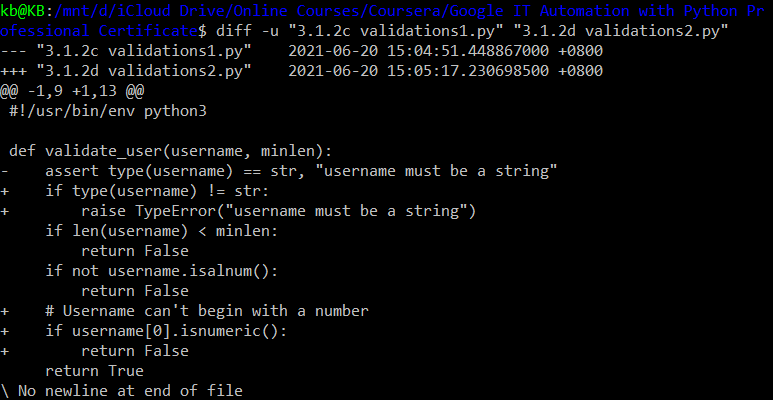
The diff tool shows all the differences between any type of file. By highlighting what’s changed, it helps us understand the changes and see how the files have been modified.



* When we call the diff command, we get only the lines that are different between two files. It's much easier to find the difference when we just have two lines.
* The less than symbol (<) tells us that the first line was removed from the first file, and the greater than symbol (>) tells us that the second line was added to the second file. In other words, the old line got replaced by the new one. In this example, we had one line that was replaced with a new one. This is a common change when modifying code, but not the only possibility.



* We can see that diff splits the changes in two separate sections.
* The section that starts with 4c4,5 shows a line in the first file that was replaced by two different lines in the second file. The number at the beginning of this section indicates the line number in the first and second files. The **‘c’** in between the numbers **means that a line was changed**.
* The section that starts with 7a9,11 shows three lines that are new in the second file. The **‘a’** stands for **added**, but that block looks a bit strange doesn't it? It seems like we're adding a return and an ‘if condition’ but no body for the ‘if block’. What's up with that? To understand this better we can use the **-u** flag to tell diff to show the differences in another format.



* This unified format is pretty different from the one that we saw before. It shows the change lines together with some context.
* Using the minus sign (-) to mark lines that were removed, and the plus sign (+) to mark lines that were added.
*  We can see that the new file actually has a completely new ‘if block’ that's part of a chain of conditionals that looks very similar, and that's why with the diff output that we saw before, it was a little confusing which lines had been added.

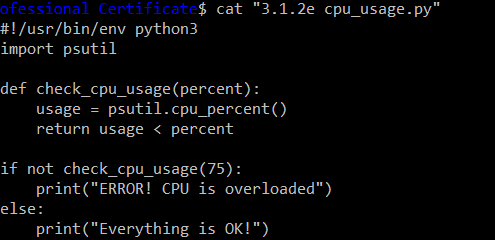
Diff is the most popular one, but not the only one available. For example, **wdiff** highlights the words that have changed in a file instead of working line by line like **diff** does. To help us even more, there are bunch of graphical tools that display files side by side and highlight the differences by using color. Some examples of this include: **meld**, **KDiff3**, or **vimdiff**. We can use these tools to give better contexts to the changes that we see.

**Applying Changes**

Imagine a colleague sends you a script with a bug and asked you to help fix the issue. Once you understood what was wrong with the script, you could describe to them what they need to change. Something like, "Well, you can only return values inside functions. I think you meant to use sys.exit instead. Also, you're converting to gigabytes twice, so your script will always fail." But this could still be hard for them to understand if the code is complex.

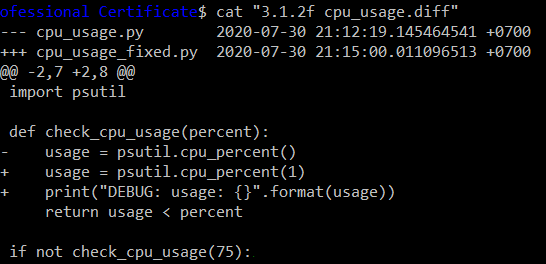
To make the change clear, you could send them a diff with the change so that they can see what the modified code looks like. To do this, we typically use a command line like **diff-u old\_file new\_ file > change.diff**. As a reminder, the greater than sign redirects the output of the diff command to a file. So with this command, we're generating a file called **change.diff** with the contents of diff-u command. By using the -u flag, we include more context which helps the person reading the file understand what's going on with the change. The generated file is usually referred to as a diff file or sometimes a patch file. It includes all the changes between the old file and the new one, plus the additional context needed to understand the changes and to apply those changes back to the original file.

Now, say you're the one receiving a diff file with a change and you want to apply it to a script you wrote. You could read the diff file you receive carefully and then manually go through the file that needs to be changed, and apply the modifications. But it sounds like a lot of manual work that could be automated, don't you think? Well, it sure is. There's a command called **patch** to do exactly this. Patch takes a file generated by diff and applies the changes to the original file.

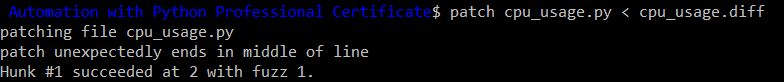


* Let's check this out in an example. Say we have a small script that checks whether the computer is under too much load, like this one.
* This script uses the psutil module to check the percentage of the CPU that's currently in use. When the load is above a threshold, in this case 75 percent, it prints a message with an error. When it's under the threshold, it says that everything's okay.

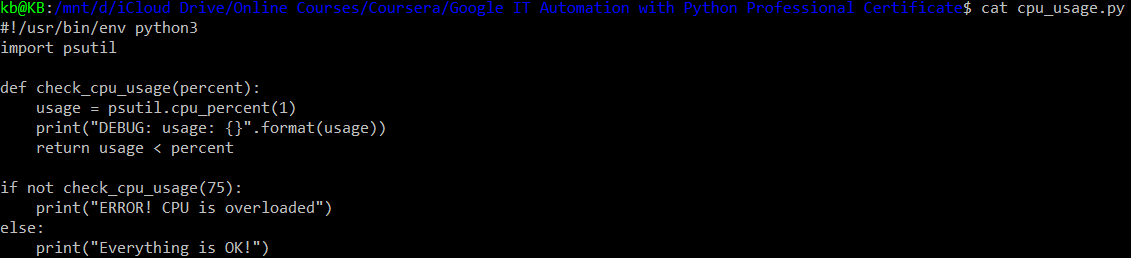
Now, we've shared this script with a few colleagues and one of them tells us that the script doesn't work correctly. Even if a computer is completely overloaded, the script will say that everything's okay. Our colleague is so helpful that they sent us a diff with the fix for our problem



* We can see that our colleague made two changes. They added a one as a parameter to the CPU percent function and they added a debugging line, that prints the value returned by the function.
* Our colleague explains that by calling the CPU percent function without a parameter, we were not averaging over a period of time, and so the call always returns zero.



* So we have the diff file and we want to apply it to our script. We'll use the patch command. We'll pass the name of the file that we want to patch in this case, cpu\_usage.py, as the first parameter to the command and then we'll provide the diff file through standard input. We will use the less than symbol to redirect the contents of the file to standard input.
* So we told patch to apply the changes that come from cpu\_usage.diff to our cpu\_usage.py file. We get one single line that says the file was patched, which means that we've successfully applied the changes.



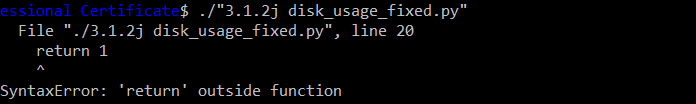
* Let's verify that by looking at the contents of our script. We see that our file was modified with the changes that we got from our colleague. The CPU percent function is being called with a parameter of one and the debugging line is printed. Once we're happy with the script, we could remove the debugging line.

Summary: While **diff** is the command that generates the difference between two files, **patch** is the command that applies those differences to the original file.

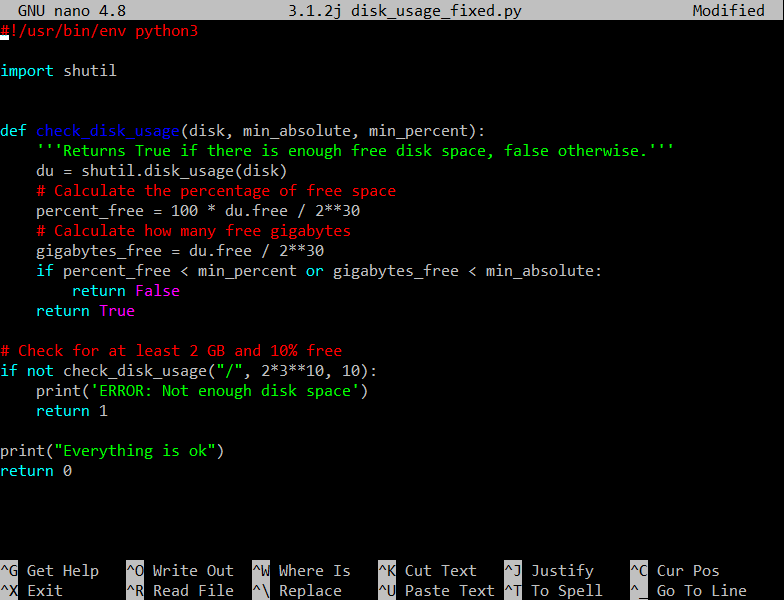
**Practical Application of diff and patch**

Imagine this, a colleague is asking our help with fixing a script named disk\_usage.py. The goal of the script is to check how much disk space is currently used, and print an error if it's too little space for normal operation. But the script is currently broken because it has a few bugs. We’ll help our colleague fix those bugs to demonstrate how to use diff and patch. Before we change anything, let’s make a couple copies of the script. We'll add \_original to one copy, which we’ll keep unmodified and use for comparison and \_fixed to the other copy, which we’ll use to repair our fix.

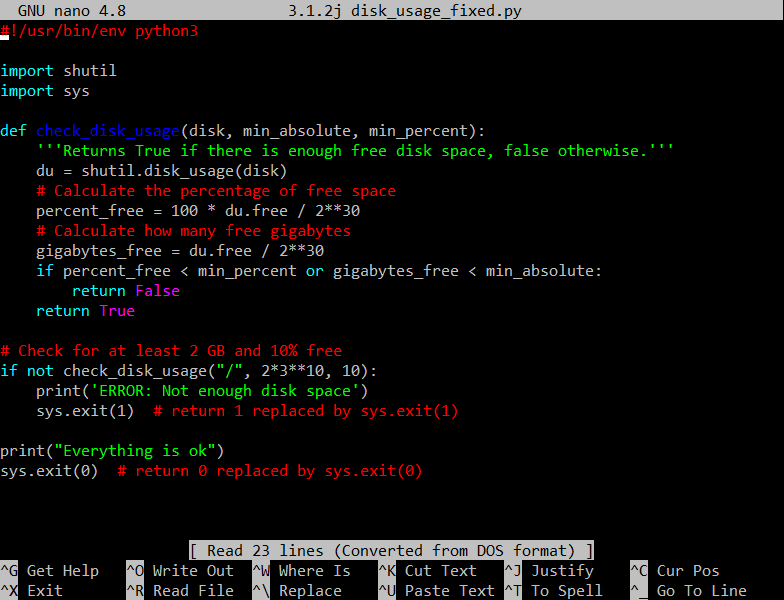




* Python is complaining that there's a return outside of function. And if we look at the code, we can clearly see that there's a return that's not inside any function.



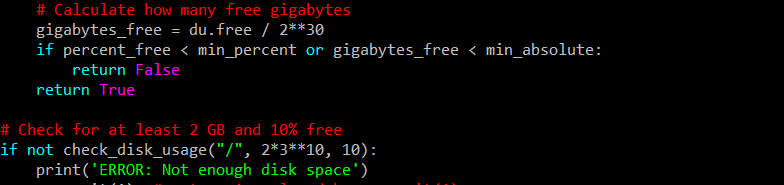
* Before modified



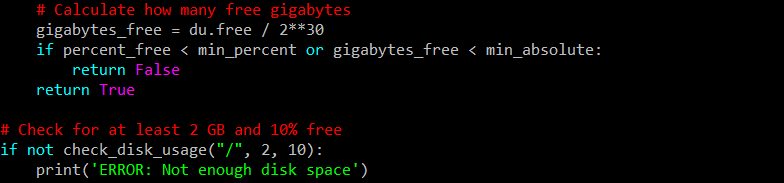
* After modified: We can only use return statements inside functions. So how do we fix this? There’re a couple options. We could turn the current code into a function and then call that function from the main part of our script. Or we could use sys.exit to make the return number of the exit code of our script, which is the code that causes a program to exit with the corresponding exit value. For now, let's go with the second option.



* Okay, we've made the change. Let's execute this new version of our script. Darn, we fixed the syntax error, but now the script is telling us we don't have enough space on our disk. But we know that we actually do have some free space, right? What's up with that?



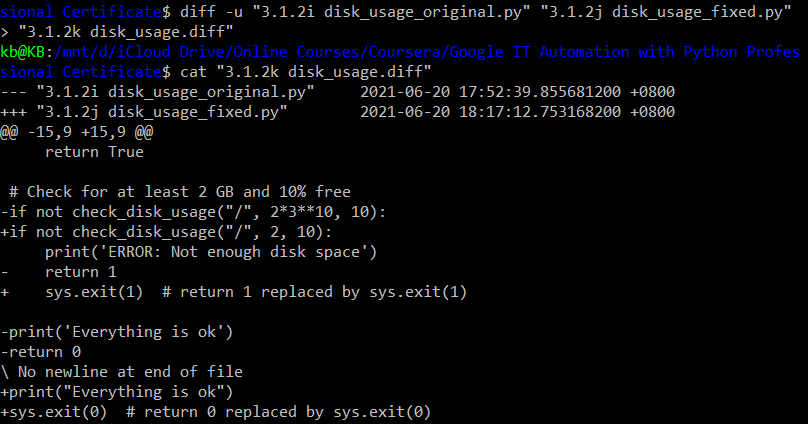
* If you look closely at the code, you might notice that the script is converting to gigabytes twice.
* The function call to check\_disk\_usage is passing 2 \* 2 \*\* 30. You might remember that the \*\* operator is used to calculate powers. In this case, 2 to the power of 30, which is how many bytes are in a gigabyte. So, this would be 2 gigabytes, but that be if the check\_disk\_usage function was expecting a value in bytes.
* If we look at the code of the function, we can see that it's already dividing the amount of free bytes by 2 to the power of 30. So in other words, we're doing the gigabyte conversion twice. Once when calling the function and once inside the function.



* We need to get rid of one of them. Let's change how we call the function.



* It works now.



* Okay, now we need to send a fixed to our colleague so that they can fix their script. To do that, we'll use a technique we just learned to generate a diff file, and then check the contents of the diff using the cat command.



* So this is what we need to send to our colleague to have them patch their file. They would run the patch command like this. By calling patch with the diff file, we've applied the changes that were necessary to fix the bugs.



* Let's check that disk\_usage.py now executes successfully. Success!

Summary: So this is how we can look at differences between files, generate diff files together to gather our changes, and then apply those changes using patch. But this is still a very manual process, where version control systems can really help.

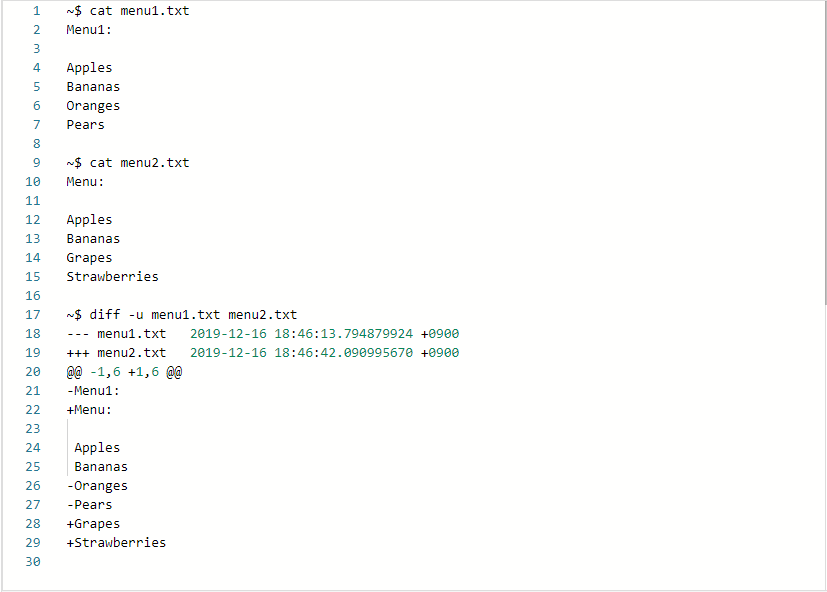
**diff and patch Cheat Sheet**

**diff**

diff is used to find differences between two files. On its own, it’s a bit hard to use; instead, use it with diff -u to find lines which differ in two files:

**diff -u**

diff -u is used to compare two files, line by line, and have the differing lines compared side-by-side in the same output. See below:



**Patch**

Patch is useful for applying file differences. See the below example, which compares two files. The comparison is saved as a .diff file, which is then patched to the original file!

