

---

# Shell Scripting Commands

Let's investigate how Copilot and ChatGPT does with shell scripting. A while back I wrote an article I called “[What the ~|&||&&\\$!`\(\)?,](https://bit.ly/what-the-symbols)” (<https://bit.ly/what-the-symbols>) mostly about what those symbols mean in a bash (or similar) shell and how to use them.

Can GitHub Copilot and ChatGPT give me these types of commands from nothing but comments? Read on to find out.

## **#!/bin/bash**

The first thing I did was to create a blank file in VS Code. At the top of the file, I added a “shebang” line to indicate that the file contains shell scripting code.

```
#!/bin/bash
```

Next, I added the following comment to see what Copilot would write for me.

```
# Check for a directory and print a message if it doesn't exist
```

To my surprise, it didn't do anything. Maybe, I thought, Copilot doesn't know about shell scripting and this experiment will end quickly. Then it hit me. It didn't get the hint from the shebang, but probably from the file extension. I saved the file as copilot.sh and tried again.

This time, Copilot gave me some code.

```
if [ ! -d /home/username/Downloads ]; then
    echo "The directory /home/username/Downloads does not exist."
fi
```

It isn't what I expected, but it does what the comment says.

---

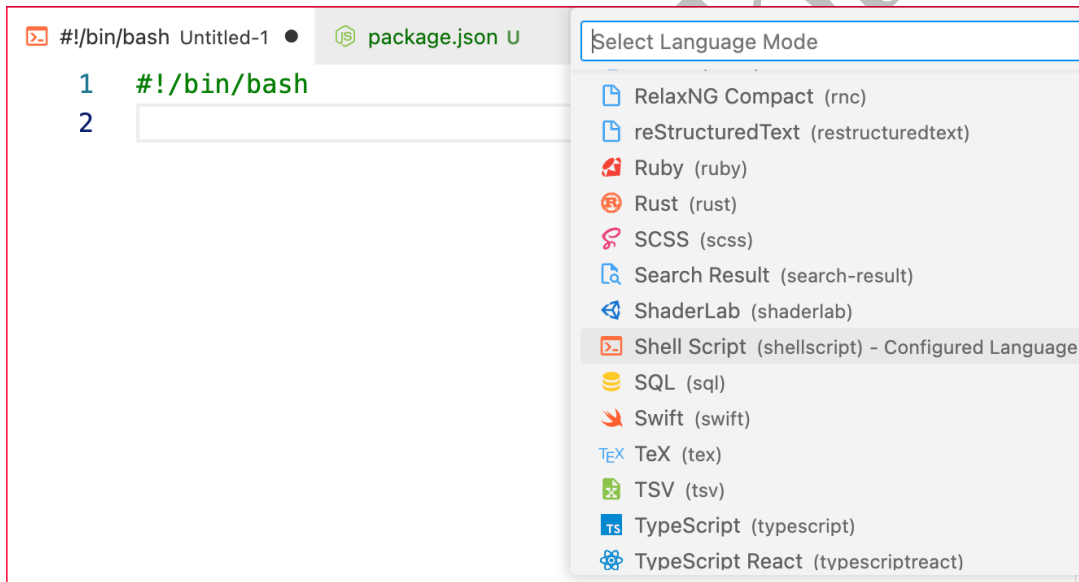
---

## File Extension?

As I mentioned, Copilot needs to know what kind of file it is before it can write code for you. Through further experimentation, I discovered that it isn't the file extension that made Copilot understand the file type. Instead, it was VS Code itself. Saving the file with a .sh extension told VS Code that it was a Shell Script file.

Ln 1, Col 1   Spaces: 2   UTF-8   LF   Plain Text

The file extension is less important than the type VS Code thinks it is. In most cases, VS Code gets it right automatically. If not, you can click on the phrase “Plain Text” in the bottom status bar. A menu will appear at the top of the page, allowing you to select the file type manually.



In my case, saving the file with a .sh did the trick. In another instance, adding the “Shebang” line did it. The results seemed inconsistent, so double-check the type.

Ln 2, Col 1   Spaces: 2   UTF-8   LF   Shell Script

The important thing to remember is that Copilot only cares about that file type and not the extension.

---

---

## What about ChatGPT?

ChatGPT seems better suited for these sorts of things, given its more conversational nature. Here is how it responded to the same prompt:

*You can use the **test** command (also known as **[** or **[[**) to check if a directory exists, and the **echo** command to print a message if it doesn't. Here is an example:*

```
if test ! -d /path/to/directory; then
    echo "Directory does not exist"
fi
```

*Alternatively, you could use the **if [ ! -d "/path/to/directory" ]; then** command with the same echo message.*

*Another option is to use **if [[ ! -d "/path/to/directory" ]]; then***

## The find Command

The next thing I tried was to try to find all PDF files anywhere in my home directory and below. In my article, I use the find command. Is that what Copilot would suggest? I entered the following comment:

```
# Find all PDF files in my home directory and print the path to each one
```

When I pressed the Return key, nothing happened. I tried a few more times, but Copilot refused to suggest anything. Then I decided to try to give it a hint. I typed the word find on a line by itself, just beneath that comment. Copilot immediately sprang to life and suggested the following completion:

```
find /home/username -name "*.pdf" -print
```

I was a little disappointed. That command won't work unless my home directory just happens to be /home/username. However, then I started thinking about it. Copilot did the hard part. It got the syntax of the find command correct. I can always edit it to fix my home directory.

In fact, when I changed my hint to find ~, it did the right thing with no further aid.

---

---

ChatGPT gave me two different options.

You can use the **find** command to search for all PDF files in your home directory and print the path to each one. Here's an example:

```
find ~ -name "*.pdf" -print
```

This will search for all files in your home directory (~) that end in **.pdf** and print the path to each one.

Alternatively, you could use **ls** command with wildcard,

```
ls ~/ -R | grep .pdf$
```

This will recursively list all files in your home directory, and filter only the files with **.pdf** extension.

As I expected, ChatGPT does a much better job than Copilot. I like that it provides an explanation and not just the command. If you are trying to learn, this is the better experience.

## Background Tasks

The next thing I wanted to do was to determine whether the AI knows how to use those symbols from my article. I changed my comment as follows, and then gave it the same hint from above.

```
# Find all PDF files below my home folder and write the results to a file called pdfs.txt in the background.
```

```
find ~ -name "*.pdf" > pdfs.txt &
```

That's exactly right, though I was confused that it dropped the **-print** parameter. Then I remembered that printing is the default behavior of the **find** command.

As before, ChatGPT gave me multiple options, complete with thorough explanations.

You can use the **find** command to search for all PDF files below your home directory, and the **tee** command to write the results to a file called **pdfs.txt** in the background. Here's an example:

```
find ~ -name "*.pdf" -print | tee pdfs.txt &
```

---

---

*This will search for all files in your home directory and its subdirectories that end in **.pdf** and print the path to each one. The output will be piped to the **tee** command, which will write it to a file called **pdfs.txt** in the current directory. The **&** at the end will run the command in background, so you can continue to use the terminal.*

*Alternatively, you could use **ls** command with wildcard,*

```
ls ~/ -R | grep .pdf$ >> pdfs.txt &
```

*This will recursively list all files in your home directory and its subdirectories, and filter only the files with **.pdf** extension, and write it to **pdfs.txt** file. The **&** at the end will run the command in background as well.*

## Output Pipe

Next, I wanted to see whether it knows how to pipe the output of one command as the input to another. I was also curious to know whether it is aware of the rest of the code in the file, as it was with TypeScript. Will it remember the earlier command?

Here is my comment and the command it generated.

```
# Count the number of PDF files in my home folder in the background and  
print the result.
```

```
find ~ -name "*.pdf" | wc -l &
```

I had hoped it would use the file it created from the last command, but it didn't. Instead, it remembered that I prefer to use **find ~**. It suggested that.

ChatGPT again gave me multiple options.

*You can use the **find** command to search for all PDF files in your home directory, and the **wc** command to count the number of files and print the result. Here's an example:*

```
find ~ -name "*.pdf" -print | wc -l > count.txt &
```

---

---

*This will search for all files in your home directory and its subdirectories that end in **.pdf**, print the path to each one, then pipe the output to the **wc** command with the **-l** option which will count the number of lines and redirect the output to a file called **count.txt** in the current directory. The **&** at the end will run the command in background, so you can continue to use the terminal.*

*Alternatively, you could use **ls** command with wildcard:*

```
ls -R ~/ | grep .pdf$ | wc -l > count.txt &
```

*This will recursively list all files in your home directory and its subdirectories, filter only the files with **.pdf** extension, pipe the output to the **wc** command with the **-l** option which will count the number of lines and redirect the output to a file called **count.txt** in the current directory. The **&** at the end will run the command in background as well.*

At this point, I found myself wondering whether it ever makes sense to use Copilot for the command line. I continued to do so for the rest of these tests.

## Redirect Errors to Ignore Them

Most people know how to use the **>** character to redirect the output of a command to a file. If the command encounters any errors, those will still display in the terminal, which can be confusing when running the command in the background.

I know it is possible to redirect errors to a special file called **/dev/null**, so that they are ignored completely. Unfortunately, I can never remember the syntax. I don't know why because it isn't that complicated. It's one of my weird blind spots, I suppose.

Fortunately, Copilot knows exactly how to do it. I decided to combine some of the above concepts to see what it would do. It got it right on the first try.

```
# Find all the PDF files below my home folder, in the background,
ignoring errors, and write the output to a file.
```

```
find ~ -name "*.pdf" > pdfs.txt 2> /dev/null &
```

---

---

What's really cool about ChatGPT is that it remembers earlier parts of your conversation. For this one, I changed up the prompt and simply asked it to "rewrite the last command to ignore any errors." The explanations are almost identical as those it provided above, so I'll spare you the duplication and simply show you the commands it gave me. Again, it gave me two options.

### Option 1 – Find

```
find ~ -name "*.pdf" -print 2> /dev/null | wc -l > count.txt &
```

### Option 2 – ls

```
ls -R ~/ 2> /dev/null | grep .pdf$ | wc -l > count.txt &
```

## \*nix System Administration

I started wondering whether they could help me with some common system administration tasks, such as creating users, adding users to groups, and so forth.

### Add a new user account

I decided to start with something simple, so I asked Copilot to create a new user account for me.

```
# Create a new user account  
  
sudo adduser username
```

This doesn't work on a Mac. Maybe it works on Linux. I decided to try again and got this.

```
# Create a new user account on macos  
  
dscl . -create /Users/username
```

---

---

That gave me a permission error. I tried once again, prepending `sudo` to the command. That one might have worked; I'm not sure. The folder `/Users/username` does not exist, nor does a user by that name appear when I opened the Users control panel. Maybe I'm still missing something.

This is an example where Copilot might be more frustrating than helpful.

ChatGPT again outdid itself, providing more detail and explanation.

*To create a new user account on macOS, you can use the `dscl` (Directory Service command line utility) command. Here is an example of how to create a new user account called "newuser" with a password of "newpassword":*

```
sudo dscl . -create /Users/newuser
sudo dscl . -create /Users/newuser UserShell /bin/bash
sudo dscl . -create /Users/newuser RealName "New User"
sudo dscl . -create /Users/newuser UniqueID "501"
sudo dscl . -create /Users/newuser PrimaryGroupID 20
sudo dscl . -create /Users/newuser NFSHomeDirectory /Users/newuser
sudo dscl . -passwd /Users/newuser newpassword
```

*This creates the account, sets the shell to `/bin/bash`, sets the `RealName`, `UniqueID`, `PrimaryGroupID` and home directory, and sets the password.*

*Alternatively, you can use the **`sysadminctl`** command to create a new user account and set the password. Here is an example:*

```
sudo sysadminctl -addUser newuser -fullName "New User" -password
newpassword -admin
```

*This creates the account with the username "newuser", full name "New User", password "newpassword" and admin privilege.*

*You also can use the GUI by going to System Preferences > Users & Groups > click on the lock icon to unlock it > click on the "+" sign to add a new user.*

*Please note that these commands require administrative privileges and should be used with caution.*

I tried the second command on my Mac and it worked flawlessly. ChatGPT wins here handily.

---



---

## Make an existing user an admin

Next, I asked them both for the command to make an existing user an admin.

```
# Make user 'michael' a local administrator  
  
sudo dseditgroup -o edit -a michael -t user admin
```

I'm honestly not sure whether that one will work, but it looks correct. Again, is Copilot being helpful or frustrating?

ChatGPT again gave me a set of options, the second of which matched Copilot's suggestion.

*To make a user "michael" a local administrator on macOS, you can use the **dscl** (Directory Service command line utility) command. Here is an example of how to make the user "michael" a local administrator:*

```
sudo dscl . -append /Groups/admin GroupMembership michael
```

*This command adds the user "michael" to the "admin" group. Members of the "admin" group have administrative privileges on the system.*

*Alternatively, you can use the **dseditgroup** command to add a user to the "admin" group. Here is an example:*

```
sudo dseditgroup -o edit -a michael -t user admin
```

*This command adds the user "michael" to the "admin" group.*

*You also can use the GUI by going to System Preferences > Users & Groups > click on the lock icon to unlock it > right click on the user you want to change > select "Advanced Options" > change the "Account type" to "Administrator".*

*Please note that these commands require administrative privileges and should be used with caution.*

## Backup Files

Have you ever used rsync to back up files from one system to another? No? Of course not.

It's hard to remember the syntax. That's why Dropbox and other such services exist.

Maybe Copilot knows the syntax.

---

```
# Sync my home folder to a remote server  
rsync -avz --delete ~/ username@server:/home/username
```

It suffers from the same path name problems from before but again, those are easily recognized and fixed. What isn't so easily discerned is whether those parameters are correct. I did a little research on my Mac using `man rsync`, to read the command's online manual. It seems that the command above is exactly what you would want to synchronize the source folder to the destination machine.

The `--delete` parameter tells the command to delete any extra files at the destination that are not on the source. If you just want to copy everything, keeping files on the destination you might have deleted locally, you can omit this parameter.

ChatGPT gave me four different suggestions, with `rsync` being the first one. It also suggested using `scp`, which I had planned to ask about next.

*There are several ways to sync your home folder to a remote server on macOS. Here are a few options:*

**rsync:** This command-line tool can be used to synchronize files and directories between two locations. Here's an example of how to use **rsync** to sync your home folder to a remote server:

```
rsync -avz -e ssh ~/ username@remote-server:/path/to/remote/folder/
```

*This command will sync all files and directories in your home folder to the specified remote server. The **-a** option tells **rsync** to preserve file permissions and ownership, the **-v** option provides verbose output, and the **-z** option enables compression to speed up the transfer. The **-e** option specifies the remote shell to use, in this case **ssh**.*

**scp:** This command-line tool can be used to copy files and directories between a local machine and a remote server. Here's an example of how to use **scp** to sync your home folder to a remote server:

```
scp -r ~/ username@remote-server:/path/to/remote/folder/
```

*This command will copy all files and directories in your home folder to the specified remote server. The **-r** option tells **scp** to copy directories recursively.*

---

*Finder: You also can use Finder to sync your home folder to a remote server by mounting the remote folder as a local volume using SMB, NFS or FTP protocols, and then use the Finder to copy the files and folders.*

*Third party software: There are also third-party software like ChronoSync, Carbon Copy Cloner, and others that can be used to sync your home folder to a remote server.*

*Please note that these commands require a remote server with the appropriate permissions and access to be set up, and also make sure you have the correct login credentials to access the remote server.*

## Securely Copy from Server to Local

Instead of backing up files, maybe there are some files on a remote server that you want to copy locally. You might use a command like this.

```
# Securely Copy from Server to Local folder  
  
scp -r username@server:/home/username/ ~/Downloads
```

Yes, that one is correct, too. ChatGPT had already suggested it as a way to sync files to a remote server, and I fully expected it to get this one with ease.

*To securely copy files from a remote server to a local folder on macOS, you can use the **scp** command. **scp** stands for "secure copy" and it uses the SSH protocol to transfer files.*

*Here's an example of how to use **scp** to copy a file called "file.txt" from the remote server "remote-server" to the local folder "~/Downloads":*

```
scp username@remote-server:/path/to/file.txt ~/Downloads
```

*This command will copy the file "file.txt" from the specified path on the remote server to the "Downloads" folder in your home directory.*

*You can also copy a directory and its contents by adding the **-r** option, which stands for "recursive"*

```
scp -r username@remote-server:/path/to/directory ~/Downloads
```

*You can also copy multiple files and directories by listing them, for example:*

```
scp username@remote-server:/path/to/file1.txt /path/to/file2.txt  
/path/to/directory1/ ~/Downloads
```

*You also can specify a different port number if the remote server uses a non-default port by using `-P` option:*

```
scp -P 22 username@remote-server:/path/to/file.txt ~/Downloads
```

*Please note that these commands require a remote server with the appropriate permissions and access to be set up, and also make sure you have the correct login credentials to access the remote server. Also, the **scp** command encrypts the data as it is transferred, providing an additional layer of security.*

## Archiving Folders

Sometimes we want to back up a folder by creating an archive. We might want to email it or upload it to a public server, or just simply copy it to a flash drive. If the goal is to keep the files together, an archive is perfect. If only I could remember what that command syntax is.

I gave them both the same prompt, with Copilot first. I asked them each for both the commands to archive and unarchive a given directory.

```
# Archive the src directory using tar and gzip  
  
tar -cvzf src.tar.gz src  
  
# Uncompress the tar.gz file  
  
tar -xvzf src.tar.gz
```

The commands are correct, but again, Copilot provides the information without context. In source code, this makes a lot of sense. If you don't know what the tar command does, though, you may still be lost. You certainly won't know that you can safely remove the 'v' from the parameter and still get the results you want. Let's see how ChatGPT did.

*To archive the **src** directory using **tar** and **gzip**, you can use the following command:*

```
tar -czvf src.tar.gz src/
```

This command creates a new archive file called **src.tar.gz** that contains the contents of the **src** directory. The **-c** option tells **tar** to create a new archive, the **-z** option tells it to use **gzip** to compress the archive, and the **-v** option tells it to display verbose information about the progress of the archive creation. The **-f** option specifies the name and location of the archive file.

You can also use **tar** with **gzip** with the following command

```
tar -zcvf src.tar.gz src/
```

It will produce the same result.

Please note that this command will create the archive file in the current directory, you can specify a different directory to save the archive file. Also, the original **src** directory will not be deleted, if you want to delete it after archiving you can add **--remove-files** option

```
tar -czvf src.tar.gz --remove-files src/
```

This command will archive the **src** directory and remove the original directory after archiving it.

This is the first time I have seen ChatGPT provide the exact same information as an alternative to its first suggestion. Notice that the first two commands are identical except for the ordering of parameters, which is irrelevant. It's literally the same command.

I do appreciate the third option it gave, though. I was not aware of the **--remove-files** option.

I then asked it to uncompress the tar file. I admit that I was not prepared for the tutorial it generated.

To uncompress a **.tar.gz** file, you can use the following command:

```
tar -xzvf src.tar.gz
```

This command tells **tar** to extract the contents of the **src.tar.gz** file, the **-x** option tells it to extract the archive, the **-z** option tells it to use **gzip** to decompress the archive, and the **-v** option tells it to display verbose information about the progress of the extraction. The **-f** option specifies the name and location of the archive file.

You can also use the following command

```
tar -xvf src.tar.gz
```

---

This command will also extract the contents of the **src.tar.gz** file, but you need to specify that the file is gzipped by adding the **-z** option

```
tar -xvzf src.tar.gz
```

This command will extract the contents of the archive to the current directory, if you want to extract the files to a different directory you can use the **-C** option

```
tar -xvzf src.tar.gz -C /path/to/directory
```

This command will extract the files to the directory **/path/to/directory**.

Please note that the extracted files will have the same permissions and ownership as the original files, if you want to change the permissions or ownership of the extracted files you can use the **--no-same-permissions** and **--no-same-owner** options.

```
tar -xvzf src.tar.gz --no-same-permissions --no-same-owner
```

This command will extract the files with the current user's permissions and ownership.

Also, after extracting the files you can delete the original archive file by adding **-k** option

```
tar -xvzf src.tar.gz -k
```

This command will extract the files and keep the original archive file.

It's important to note that tar.gz files are also known as tarballs, they are a combination of tar archive and gzip compression, the **.tar** file contains the files and the **.gz** file is the compression. It's important to use both options **-x** and **-z** to extract and decompress the file respectively.

Once again, it managed to get confused and essentially gave me the same command for two different options. Otherwise, I think it did a pretty thorough job explaining things.

## GitHub SSH Key

I don't know about you, but I always have to look up how to create a new SSH key for GitHub using git+ssh protocol. Maybe I could get Copilot to do it for me.

```
# Create and copy a new ssh key for github to the clipboard
```

---

```
ssh-keygen -t rsa -b 4096 -C "  
" && cat ~/.ssh/id_rsa.pub | pbcopy
```

This command is correct and exactly what I expected, with the exception of the -C " prompt it gave me. In fact, Copilot stopped after that parameter, as if it expected me to input something.

Whenever Copilot gives you a partial command like that, it is either waiting for you to enter more info yourself or it wants you to continue on the next line.

As it turns out -C is simply a comment to the key, so I could have typed something like github-key and continued on. As soon as I pressed Return, it gave me the second line. This line tells the shell to copy the public key from my home directory's .ssh directory into the clipboard using pbcopy.

At that point, the computer's clipboard should have a copy of my public key, which I can then paste into GitHub's public key UI.

ChatGPT suggested that I use my email address as the comment and gave me a complete example. It also split the process into two commands instead a one.

*To create and copy a new SSH key for use with GitHub on macOS, you can use the **ssh-keygen** and **pbcopy** command. Here is an example of how to create a new SSH key and copy it to the clipboard:*

*1. Open the Terminal and generate a new SSH key:*

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

*This command creates a new RSA key with a 4096-bit key size, and associates the email "**your\_email@example.com**" with the key as a comment.*

*2. When prompted, you can press Enter to accept the default location and file name for the key, or enter a new location and file name.*

*3. After the key is generated, you can copy the key to your clipboard using the **pbcopy** command:*

```
pbcopy < ~/.ssh/id_rsa.pub
```

*This command copies the contents of the file "~/.ssh/id\_rsa.pub" to the clipboard.*

---

4. Go to your GitHub settings, under SSH and GPG keys, click on "New SSH Key" button.

5. Give a title to the key, and paste the key on the "Key" field, press "Add SSH key" button.

6. You can test the connection by running **ssh -T git@github.com**

Please note that **pbcopy** command only works on macOS and is used to copy data to the clipboard. On other operating systems, the command may differ and you may need to use **xclip** or **xsel** to copy the key to the clipboard.

I love that ChatGPT goes above and beyond the actual command, providing examples, options, and even caveats.

## Commands Specific to macOS

So far, I've been impressed with what both Copilot and ChatGPT were able to do. There are other commands that I have a hard time remembering, particularly commands on macOS that manipulate media files. Let's see how they did with some of those.

Unfortunately, I seem to have hit the limit of Copilot's abilities. Each of the comments I tried did not result in working commands. Here is the first, asking Copilot to convert a folder of JPEG files to PNG format.

The command it gave me doesn't work on my Mac, but it looks like it should.

```
# Convert all jpg files in the current directory to png files

for file in *.jpg; do
    convert "$file" "${file%.jpg}.png"
done
```

ChatGPT, naturally, explained why.

You can use the command-line tool **convert** from ImageMagick to convert all JPG files in the current directory to PNG files. Here is an example of how to use **convert** to convert all JPG files to PNG files:

```
for file in *.jpg; do convert "$file" "${file%.jpg}.png"; done
```

---



---

*This command will loop through all files in the current directory that have a **.jpg** extension, run the **convert** command on each file, and save the output to a new file with the same name but with a **.png** extension.*

*Alternatively, you can use **find** and **mogrify** command from ImageMagick:*

```
find . -name "*.jpg" -exec mogrify -format png {} +
```

*This command will search for all **.jpg** files in the current directory and its subdirectories and change their format to **.png** and overwrite the original files.*

*Please note that this will overwrite the original JPG files. If you want to keep the original files, you can modify the command to save the output to a different location or with a different file name.*

Interestingly, its first suggestion matched Copilot's identically. The difference is that ChatGPT told me that the **convert** command comes from ImageMagick. Deciding not to give up just yet, I asked it how to install ImageMagick.

This is what it told me.

*To install ImageMagick on macOS, you can use the package manager Homebrew. Here are the steps to install ImageMagick using Homebrew:*

*1. Install Homebrew if you haven't already by running the following command in the terminal:*

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

*2. Once Homebrew is installed, you can use it to install ImageMagick by running the following command in the terminal:*

```
brew install imagemagick
```

*3. After the installation is complete, you can check that it was successful by running the following command:*

```
convert -version
```

*This will display the version of ImageMagick that you have installed.*

---

Another way to install ImageMagick on macOS is to download the binary package from the official website and install it manually. You can find the binary package here:

<https://imagemagick.org/script/download.php#macosx>

Please note that you may need to add the ImageMagick binaries to your system's PATH for the **convert** and **mogrify** command to be recognized by the terminal.

I had similar issues with other commands. This one also didn't work for me, but there was no hint as to what the error might be.

```
# Extract audio from all the mp4 files in the folder

for file in *.mp4; do
    ffmpeg -i "$file" -vn -acodec copy "${file%.mp4}.mp3"
done
```

As expected, ChatGPT gave me a lot more context, after suggesting a very similar command, but also explaining what I needed to understand about it.

You can use the command-line tool **ffmpeg** to extract the audio from all MP4 files in a folder on macOS. Here is an example of how to use **ffmpeg** to extract the audio from all MP4 files in the current directory and save it as an MP3 file:

```
for file in *.mp4; do ffmpeg -i "$file" -vn -acodec libmp3lame -ac 2 -q:a 4 -y "${file%.mp4}.mp3"; done
```

This command will loop through all files in the current directory that have a **.mp4** extension, run the **ffmpeg** command on each file, and save the audio output to a new file with the same name but with a **.mp3** extension.

This command uses the **-i** option to specify the input file, the **-vn** option to disable the video stream, the **-acodec** option to specify the audio codec, **-ac 2** to specify the number of audio channels, **-q:a 4** to specify the audio quality and **-y** to overwriting output files.

You can change the output file format and codec by changing the **mp3** and **libmp3lame** accordingly.

Please note that you may need to install **ffmpeg** in your system if it is not already installed. You can install **ffmpeg** using Homebrew by running the following command in the terminal:

```
brew install ffmpeg
```

---

*Also, please note that this command will overwrite any existing files with the same name. If you want to keep the original files, you can modify the command to save the output to a different location or with a different file name.*

The command ChatGPT gave me was more complete, but its explanation is misleading. Or maybe I simply read it wrong. The `-y` parameter will automatically overwrite any existing files on output, but the command is not destructive to the input files.

My final test was to try to extract audio using `avconvert`, another command on macOS. Copilot gave me a single command.

```
# Extract audio from video file using avconvert  
  
avconvert -i video.mp4 -vn -acodec copy audio.mp3
```

Apparently, this command no longer exists, which ChatGPT was happy to explain to me when I asked about it.

**`avconv`** is a command-line tool similar to **`ffmpeg`** that can be used to convert multimedia files. To extract the audio from a video file using **`avconv`** on macOS, you can use the following command:

```
avconv -i inputfile.mp4 -vn -acodec libmp3lame -ac 2 -q:a 4 -y  
outputfile.mp3
```

This command uses the `-i` option to specify the input file, the `-vn` option to disable the video stream, the `-acodec` option to specify the audio codec as **`libmp3lame`**, the `-ac 2` option to specify the number of audio channels, the `-q:a 4` option to specify the audio quality and the `-y` option to overwrite output files if they already exist.

You can change the input and output file name and format accordingly, also you can change the codec and quality options as you need.

Please note that you may need to install **`avconv`** in your system if it is not already installed. You can install **`avconv`** using Homebrew by running the following command in the terminal:

```
brew install libav
```

---

---

Please note that **avconv** and **ffmpeg** are similar but not exactly the same, and their syntax and options may differ slightly. Some options or codecs that work with **ffmpeg** may not work with **avconv**, and vice versa. Also, **ffmpeg** is more commonly used and has more active development than **avconv**, but both of them can be used to extract audio from a video file.

Confused and mildly curious, I then asked, “what is the difference between avconv and avconvert?”

**avconv** and **avconvert** are actually the same thing, **avconv** is the command-line tool for the **libav** library, that is a fork of the **ffmpeg** library. **avconv** was the command-line tool for the **libav** library and it's the equivalent of the **ffmpeg** command-line tool for the **ffmpeg** library, but the **libav** project was discontinued in 2018 and now the **libav** developers recommend using **ffmpeg** instead of **avconv**. **avconvert** is a typo or a misspelling of **avconv**, and both commands can be used to perform the same tasks. The **avconv** tool has been replaced by **ffmpeg**, but you might find some old resources or scripts that use **avconv** or **avconvert**.

In summary, **avconv** and **avconvert** are the same command-line tool, but the **libav** project was discontinued in 2018 and now it's recommended to use **ffmpeg** instead of **avconv**.

I got none of that context from Copilot. What amazed me is that I thought avconvert was the current and correct command to use. Apparently, I was mistaken on both counts.

## Parting Thoughts

The output from ChatGPT is superior to that of Copilot, but I expected that for the purposes of this chapter. ChatGPT is an advanced AI model that is capable of understanding and responding to more complex and nuanced questions. Copilot is an AI assistant designed to help developers with their coding tasks, but it's not as advanced as ChatGPT. However, the important thing here is that either one can help you accomplish your task effectively. Both ChatGPT and Copilot can provide accurate and reliable answers to your questions, but ChatGPT's output may be more detailed and comprehensive. Ultimately, the choice of which tool to use will depend on your specific needs and preferences.

---

---

In addition to the difference in output quality, another aspect to consider is the area of expertise of each tool. ChatGPT is a general-purpose language model that has been trained on a vast amount of text data, making it a good choice when the subject matter may be unfamiliar, or when you need an explanation or clarification. It is good at providing a comprehensive understanding of a topic, and it can be used for a wide range of tasks, including language translation, text generation, and question answering.

On the other hand, Copilot is an AI assistant that is specifically designed for developers and is trained on actual software. Therefore, may be better at understanding your code in context and providing solutions that are tailored to your specific needs. As we will see, it can help you with your coding tasks by suggesting snippets of code, providing documentation, and even completing your code for you. If you are a developer and you need help with your code, Copilot is a great choice.

Draft - Not For Sale