ZOAU1

# Z Open Automation Utilities

Deep Z functionality right from your Shell

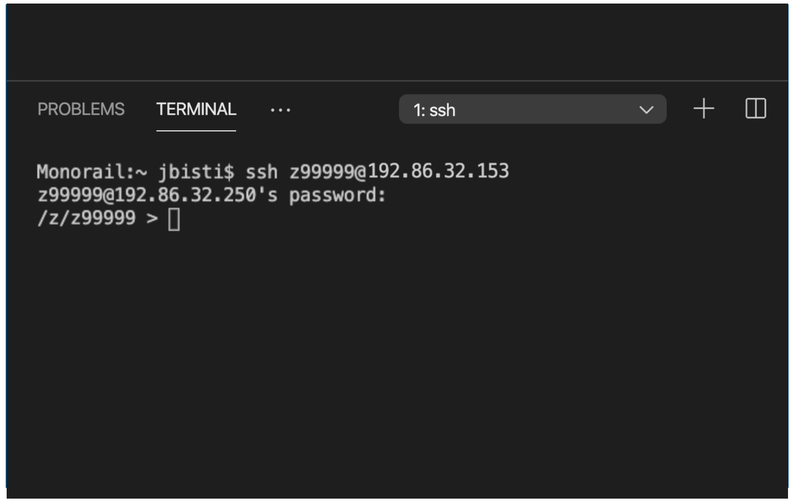
12 steps  2 hours

## THE CHALLENGE

One of the newest innovations in z/OS is the introduction of IBM Z Open Automation Utilities (ZOA Utilities). You've seen how JCL can be written to submit jobs and call programs, and even if it made sense to you, many developers have asked for a way to interact with tasks on Z through scripting. ZOA Utilities answers that, letting you perform many tasks on z/OS without needing to get into JCL.

## BEFORE YOU BEGIN

By now, you should be fairly familiar with the basic concepts of jobs, JCL, and USS on z/OS. Here, you'll be using the Z Open Automation Utilities to help streamline a lot of those processes.




## 1. OPEN UP A TERMINAL

If you are already logged into the system, great. Stay there.

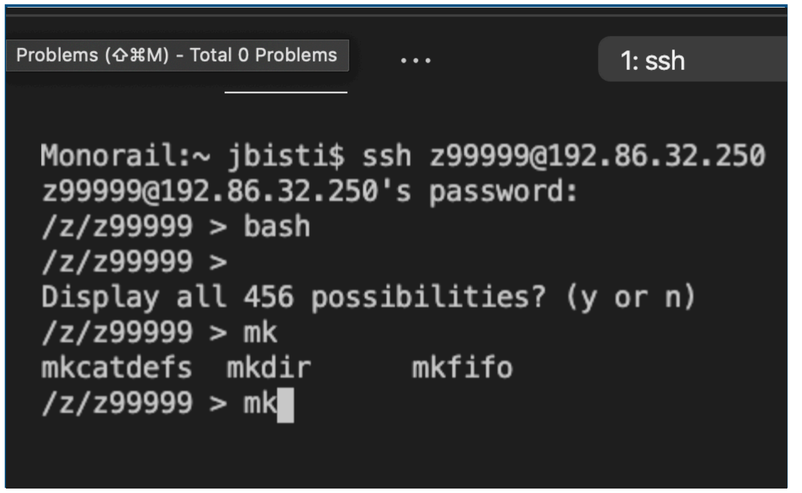
If not, open up a new Terminal connection and log in, just as you did for the USS portion of the challenges.


## 2. JUMP INTO THE BASH SHELL

Type **bash** and hit enter. That's it. Now you're in Bash.

You've just changed your shell for this login session. Now, instead of using the regular default 'sh' shell, you're using a slightly more fancy shell called the Bourne-Again SHell, shortened to 'bash'

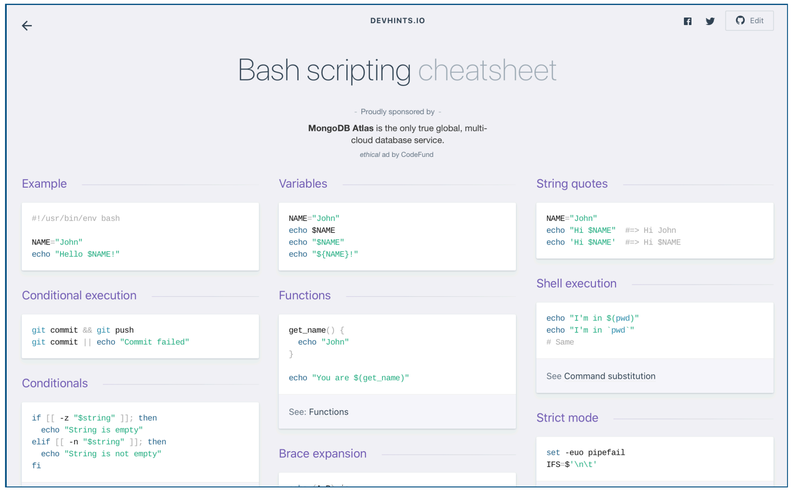
Bash is nice. It doesn't have a lot of the issues you may have run into with the regular shell, but we wanted you to learn both ways just in case you have to work on a system with a basic USS shell some time.

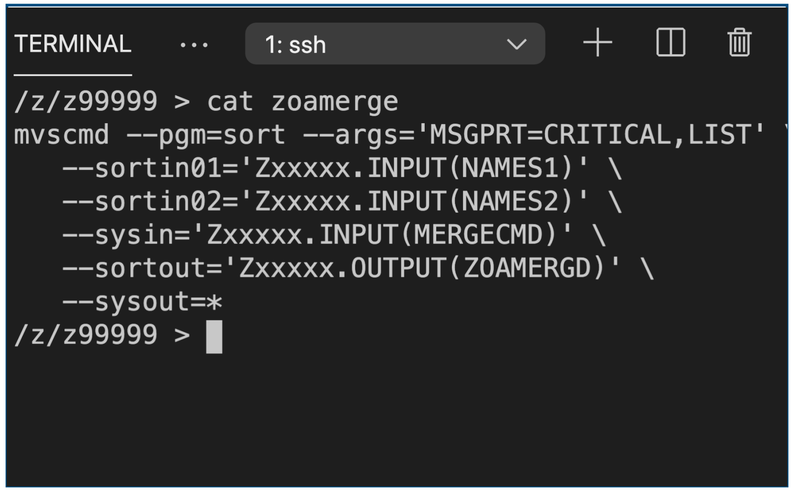


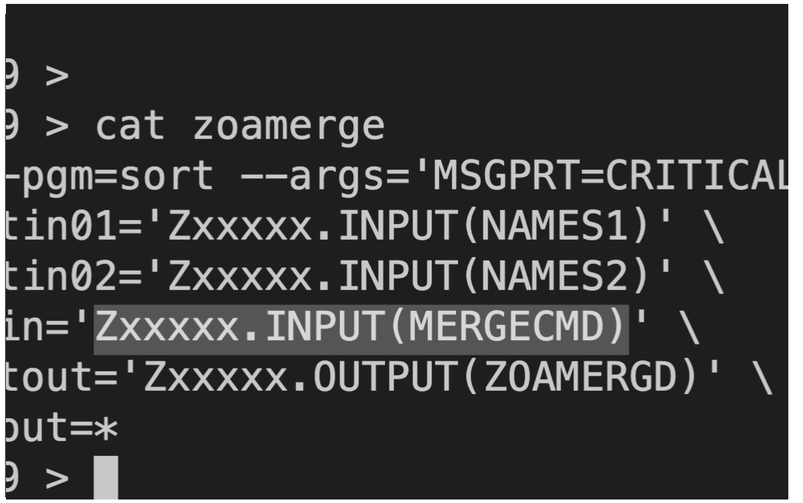

## 3. EXPLORE THE BASH SHELL

Hit the tab key a few times. It will ask if you're sure you want to see all 345 possible commands (no). Type the letters "***mk***" and then hit tab twice. It shows you the three commands that start with that letter. (If it doesn't, try hitting the tab key a third time)

This auto-complete function can be used for commands, as well as file and folder names. So if you needed to cd into a folder with a big long complicated name like mrv-4.6.293-final, you could just type cd mrv, then hit tab, and it would autocomplete.




## 4. RECALL PREVIOUS COMMANDS

Press the up arrow on your keyboard. Hit it a couple more times. You should see commands that you entered before. The Bash shell lets you easily recall previous commands. Press down to go back.

Visit https://devhints.io/bash for a more exhaustive Bash Scripting Cheatsheet. You'll want to pay special attention to the **Substitution**, **Loops**, and **File Conditions** sections.

## 5. OPEN zoamerge

Go into USS and look for a file called **zoamerge**. You can do this by typing '**cat zoamerge**' in an ssh shell or by using the USS view in VS Code. You'll see the start of some code here that won't work until you replace the Zxxxxx with your userid.

Go ahead and make those changes, but don't run it yet.

## 6. EXAMINE THE COMMAND

Look at the --sysin line of the script. The program we're running (sort) needs some additional input so it knows *how* to sort those input files.

The MERGECMD member contains the parameters for that sort command.

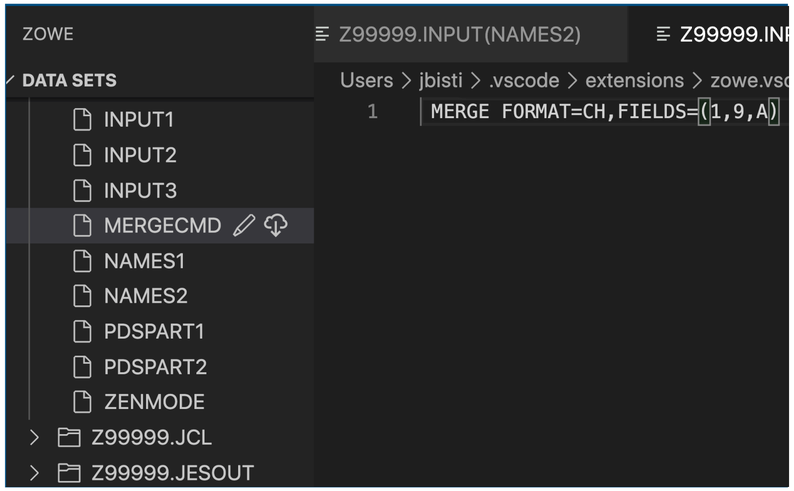
You may wonder what the reasoning is for breaking this command out into its own file, but keeping tasks and input and output separate, and then using the JCL (or in this case, the ZOAU script) to orchestrate how they all come together is a common technique to ensure consistent results.

Open up that command file and see what it looks like.

### PUNCTUATE YOUR ESCAPE (or escape your punctuation)

When you reference a data set member on z/OS, you typically reference it like this: **DATA.SET.NAME(MEMBER).** This works fine when you're in a z/OS tool, but the Bash shell looks at things a little differently, and interprets parenthesis as something else. For that reason, we put the data set and member name in single quotes so it doesn't try to interpret them. Simple as that.

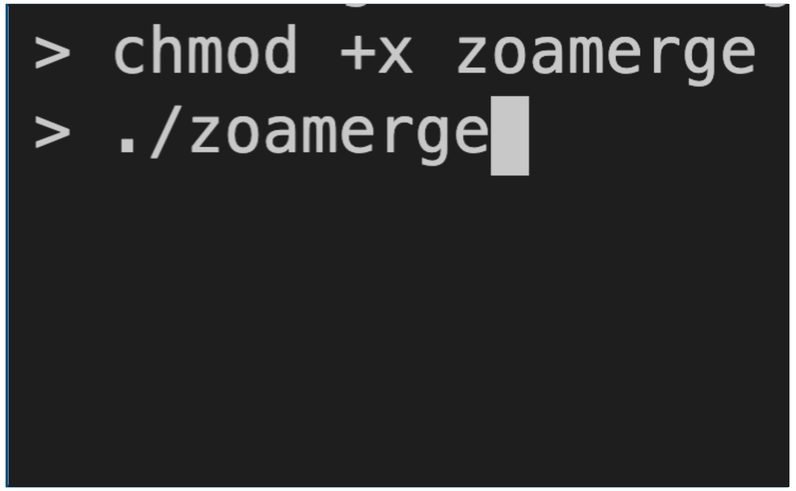
The backslashes ( \ ) say "This is actually one big command, but I want to continue it on the next line". It helps for readability, since that one mvscmd command can get pretty long, and we want the parameters to line up.




## 7. OPEN UP THE COMMAND FILE

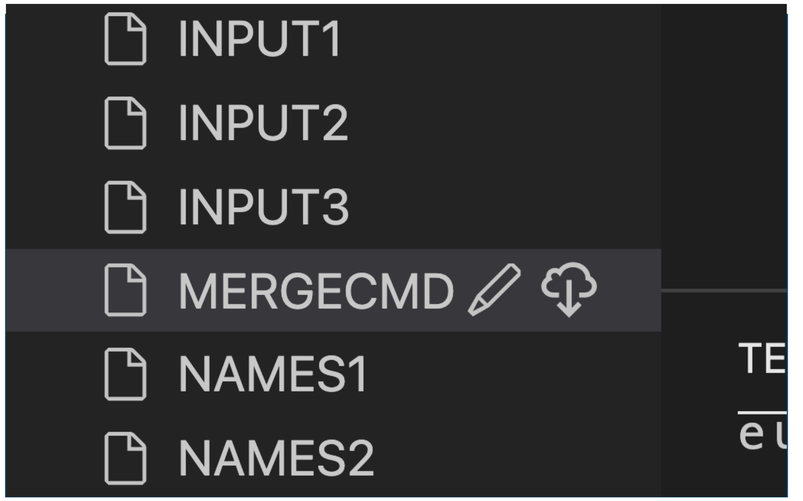
Look at the data set member referenced in the --sysin line.

It looks like it's saying we want a MERGE of the input.
**FORMAT=CH** says we're working with characters (as opposed to BInary or other formats)
**FIELDS=(1,9,A)** tells it we want to sort using the entries starting in column 1, with a length of 9, in Ascending order. It is assumed that both input files are already sorted this way, otherwise the SORT/MERGE will fail.




## 8. RUN THAT COMMAND

Make the script executeable (**chmod +x zoamerge**) and then run it with ./zoamerge. It will likely spin for a while and issue an error (unless you spotted it ahead of time).




## 9. FIX IT UP

Obviously, there's an error that you need to fix. There's a big hint in the screenshot above as well as in the bottom left corner of this page. It's a very simple fix, but it might take you a while to spot it.

**TIP**: If an empty ZOAMERGD file got created from a previous attempt, you will want to delete this before running it again.

### "OK SO, I FOUND THE ERROR. WHAT'S THAT ALL ABOUT?"

If you search on ICE005A, you'll likely find some documentation saying what's wrong. You may be thinking the SORT command is just being difficult. To be honest, it stumped us for a while as well, which is why we thought it'd make a good challenge.

Mainframe operations thrive on efficiency. Using the position of certain text is a very simple and fast way of denoting what's a command, what are parameters, and what's data. You'll see this in other languages and utilities, which is why paying attention to column markers is important.

Fortunately, whenever you run into a problem with its own error number like that, the answer is usually spelled out for you in fairly simple language.

```
./zoamerge
MERGE FORMAT=CH,FIELDS=(1,9,A)
```

## 10. RUN THE SCRIPT (AGAIN)

After fixing the problem, run the script. When it encounters zero errors, it simply echoes out the parameters sent to the SORT command and finishes. You can verify that it actually combined those two files by looking in the output file.

So, we know that it worked... but it sure would be nice if it actually said what it did, wouldn't it? Let's fix that in the next two steps.

```
/z/z99999 > cat successtest
if
 rm an_unimportant_file;
then echo "It worked!"
else
echo "It didn't work"
fi
```

## 11. GIVE SOME FEEDBACK

Take a look at the **successtest** file in your home directory. It shows an example of a script that tries to delete a file. It will give one message if the command succeeded, and another if it failed. So, not very exciting, but it's nice how it spells out whether it was successful, isn't it? Let's take a lesson from it.

Study the syntax used here and implement it in the zoamerge script so it provides similar feedback when running the sort command.

```
./zoamerge
MERGE FORMAT=CH,FIELDS=(1,9,A)

It worked!
```

## 12. BRING IT ALL TOGETHER

When completed, you should be able to run the script and get a successful message if the merge happened, and a different message if something went wrong. A successful outcome will look similar to the above screenshot. You can make the error message as helpful as you'd like, but there needs to be two different outcomes based on the success or failure of the command in your **zoamerge** file.

All good? Submit the **CHK2** job in MTM2020.PUBLIC.JCL

### NICE JOB! LET'S RECAP

You're combining several very important pieces of technology; z/OS commands, shell scripting, Z Open Automation Utilities, and your own creativity. Being able to understand the task, find and fix the problems, and create new features based on what you learned is a HUGE part of working on these systems that literally run the world. All of this will come in handy later on, so don't lose sight!

### NEXT UP...

Guess what we suggest after ZOAU1? That's right! In ZOAU2, you'll turbocharge your ZOAU1 work with Python and some pretty clever data set tricks. Get ready, we're almost there.