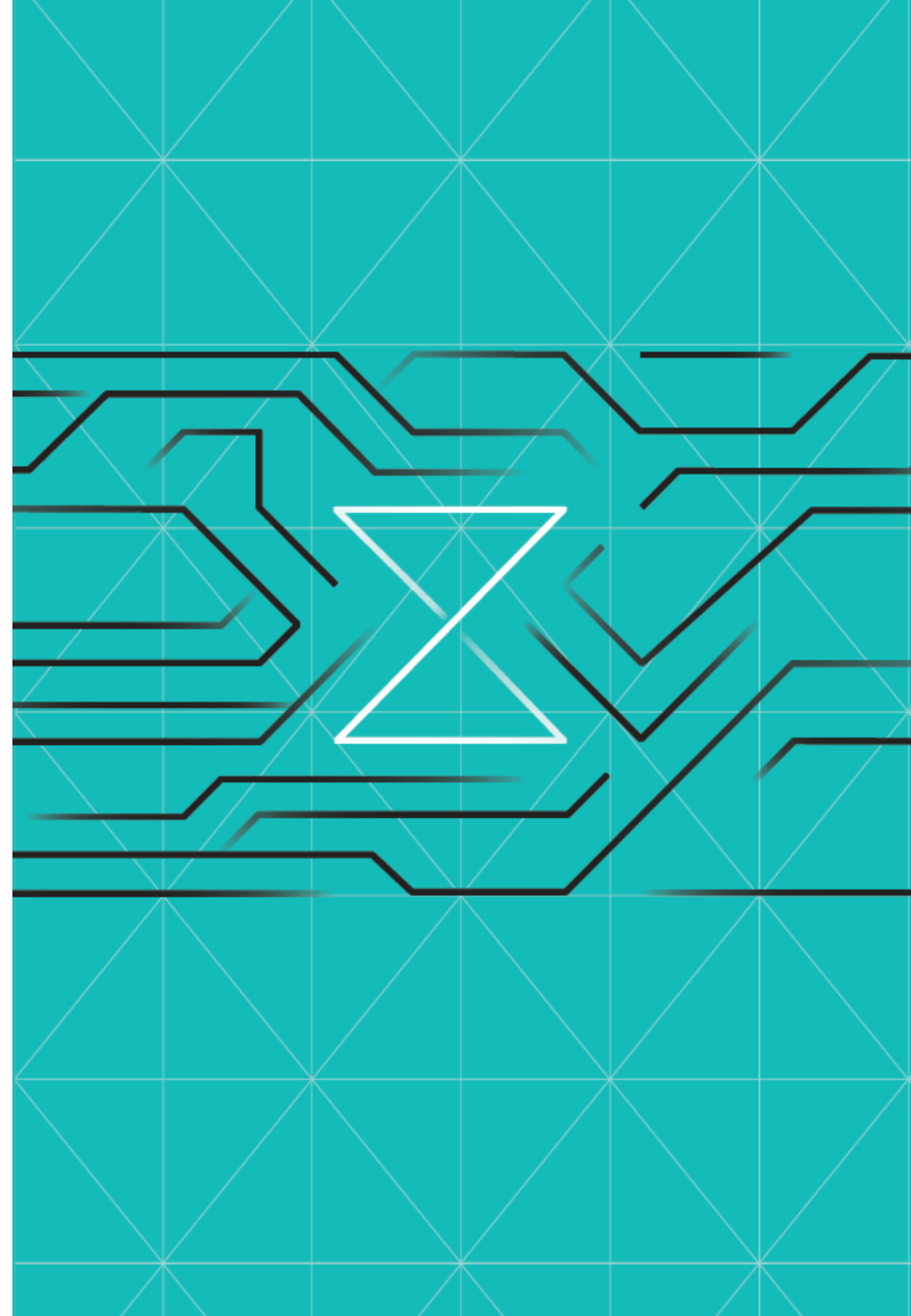


USS2

IT'S UNIX. I KNOW THIS!

- SOME SCRIPTING, SOME RUNNING, SOME CHMOD'ING
- 1 A REFRESHER
- 2 PUT IT ALL TOGETHER
- 3 FIND AND EDIT ANIMALS.SH
- 4 WHAT'S GOING ON HERE?
- 5 FIX UP THE INPUT
- 6 MAKE THE SCRIPT RUN
- 7 RUN THE SCRIPT
- 8 VERIFY COMPLETION



SOME SCRIPTING, SOME RUNNING, SOME CHMOD'ING

The Challenge

There are ways to edit files through the terminal, but for most people, it's far easier and intuitive to use the text editor built into VSCode.

For this challenge, you are going to look at USS using VSCode, work on a shell script, and make it run correctly in order to mark this challenge as complete.

Before You Begin

You should have VSCode set up and have completed the first USS challenge in Fundamentals before starting this.

Other than that, nothing else is required, you'll be entirely on the USS side of things.

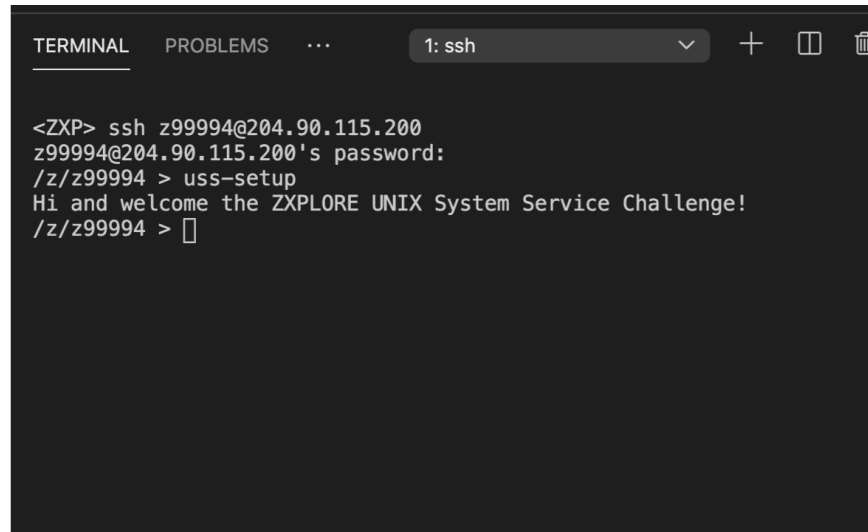
Investment

Steps	Duration
8	30 minutes

1 A REFRESHER

Before going full steam ahead into the USS2 challenge, take a moment to remember everything you have learned.

First, make sure your terminal is open, **connected through ssh** and set up correctly.



```
TERMINAL  PROBLEMS  ...  1: ssh  +  [ ]  [X]

<ZXP> ssh z99994@204.90.115.200
z99994@204.90.115.200's password:
/z/z99994 > uss-setup
Hi and welcome the ZXPLRE UNIX System Service Challenge!
/z/z99994 > 
```

If you don't remember how to do this or if your default directory is empty, take a look at the screenshot above to find the the connection and *uss-setup* from the USS Fundamentals challenge.

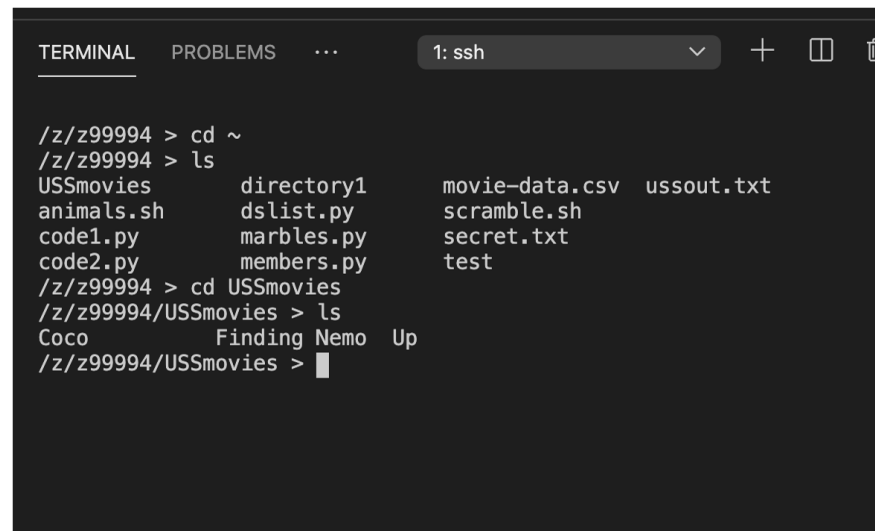
You know how to make a directory (*mkdir*), make a file (*touch*), change directories (*cd*), look around (*ls*) and displaying (print) the working directory (*pwd*).

2 PUT IT ALL TOGETHER

A quick practice of creating a directory and putting files within it.

Move back to your home directory in the terminal (/z/zxxxxx) and create a directory called **USSmovies**.

Create three files in USSmovies named after your favorite movies.

A terminal window with a dark background and light text. The window has tabs at the top: 'TERMINAL', 'PROBLEMS', and a dropdown menu showing '1: ssh'. The terminal output shows the following commands and results:

```
/z/z99994 > cd ~  
/z/z99994 > ls  
USSmovies      directory1      movie-data.csv  ussout.txt  
animals.sh      dslist.py      scramble.sh  
code1.py        marbles.py     secret.txt  
code2.py        members.py     test  
/z/z99994 > cd USSmovies  
/z/z99994/USSmovies > ls  
Coco           Finding Nemo    Up  
/z/z99994/USSmovies > |
```

Remember: If the name of your file has spaces, you'll have to put it in quotes, for example:
`touch "mac and cheese"`.

If this felt unfamiliar, keep practicing. You can continue adding files to your home directory until you become comfortable with the shortcuts and structure.

If you successfully created the directory and files, now it is time to delete them. Do you remember how to do this?

Once you have successfully removed the information you just created, you should no longer see *USSmovies* in the listing of your home directory.

WHAT AM I LOOKING AT?

When you look in a directory (or folder, if you're a Windows fan), you need a handy tool to identify what is in the directory, and what you might be able to DO with the contents (entries).

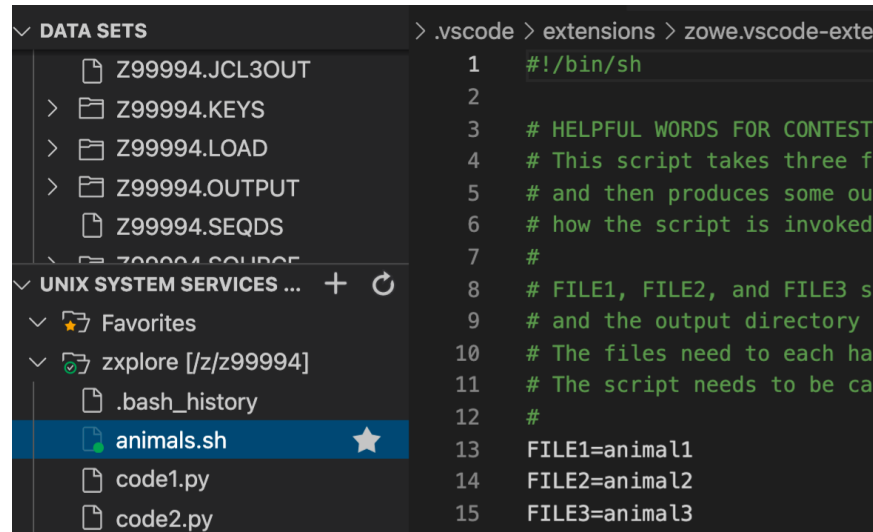
The `ls` command in its simplest form will show you the names of (unhidden) files in the directory, but it has some options commonly used to show more details (check out `man ls` for much more information):

- l shows the owner information, last update timestamp, and permissions
- a shows all the files, including hidden files (in Unix, filenames that start with "." aren't usually listed by default)
- t sorts the directory entries by last modified time
- r sorts the directory entry listing in reverse
- F adds a character code after the entry name to show if the file is "special"

3 FIND AND EDIT ANIMALS.SH

Look in your home directory and try to find **animals.sh**. This should have been copied over in the USS Fundamentals challenge, when you ran the script *uss-setup*.

If you don't see it, you can copy it from **/z/public** (or run the *uss-setup* script again).



The screenshot shows the IBM Z Xplore interface. On the left, the 'DATA SETS' panel is expanded, showing a list of files and folders. The file 'animals.sh' is highlighted. On the right, the code editor shows the contents of 'animals.sh', which is a shell script. The script starts with a shebang line '#!/bin/sh' and contains several comments and variable assignments.

```
1 #!/bin/sh
2
3 # HELPFUL WORDS FOR CONTEST
4 # This script takes three f
5 # and then produces some ou
6 # how the script is invoked
7 #
8 # FILE1, FILE2, and FILE3 s
9 # and the output directory
10 # The files need to each ha
11 # The script needs to be ca
12 #
13 FILE1=animal1
14 FILE2=animal2
15 FILE3=animal3
```

This is a script, except you are not just going to run this immediately; you are going to fix it and make it work.

4 WHAT'S GOING ON HERE?

Open up the shell script and look around. There is a large portion of the script that lives within an “IF” statement, and a lot of it will only run correctly if certain conditions are met.

```
echo "Checking for the files and folders"
#If there's three files that aren't empty (-s) AND (&&) the directory exist (-d), then do this"
if [ -s "$FILE1" ] && [ -s "$FILE2" ] && [ -s "$FILE3" ] && [ -d "$DIRECTORY1" ]; then
    echo "Everything looks good"

    #Copy the three files into that directory
    cp $FILE1 $FILE2 $FILE3 $DIRECTORY1
    echo "Files have been copied!"

    #Produce this output and put it in the "message" file
    echo "We're extremely happy to have $USERNAME on the system" > "$DIRECTORY1"/message
    MODELTYPE=$(uname -m)
    OS=$(uname -I)
    echo "The operating system is $OS running on a model type $MODELTYPE" >> "$DIRECTORY1"/message
    echo "" >> "$DIRECTORY1"/message
    echo "If $USERNAME looks out the window, they might say:" >> "$DIRECTORY1"/message

    #For every file (there should be three), do this
    for file in "$DIRECTORY1"/animal*
    do
        CURRENT_ANIMAL=$(cat $file)
        echo "Why hello there, $CURRENT_ANIMAL" >> "$DIRECTORY1"/message
    done
```

Read the comments for additional information and try to figure out what is going on in the script before proceeding.

5 FIX UP THE INPUT

The script has lots of clues about what it requires to run.

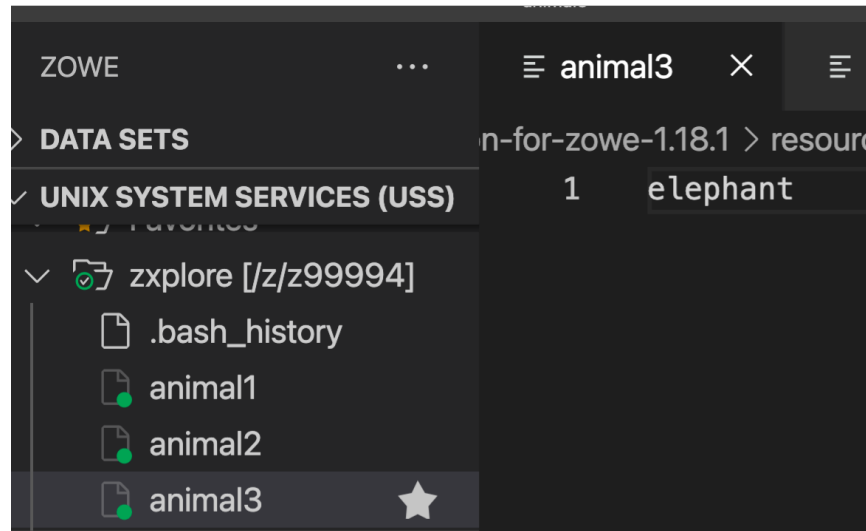
Create three files (*animal1*, *animal2*, and *animal3*) and a directory below your home directory called **uss2output**

Use VSCode to open each of these files and enter a single line into each of them, simply naming an animal you are likely to see in your area.

Don't think too hard on it, but make sure to put something in there otherwise the script can't work.

Remember to complete each line with an Enter or Return so it is a "line", not just a "word"!

(As a simple test, `cat animal[123]` should show 3 lines)



Observe the screenshot above. The file *animal1* needs to have an animal named in it. The same is true for *animal2* and *animal3*

As you work between the terminal and the GUI, there may be moments when the views become inconsistent.

Remember that you may occasionally need to reload the VSCode view to catch the latest updates.

"DOES THE WORLD NEED YET ANOTHER SCRIPTING LANGUAGE?"

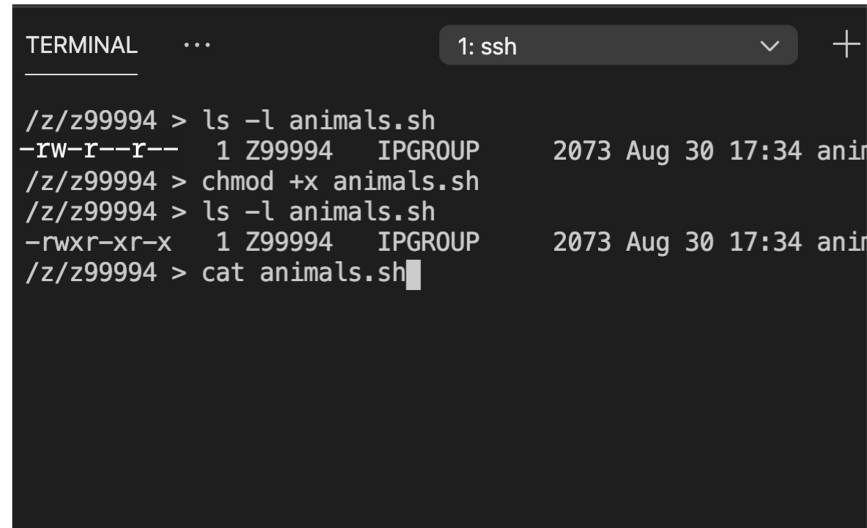
A growing number of programs running on Z came over from Linux or other UNIX platforms, where shell scripts are used to simply and transparently perform setup steps, like verifying there is enough disk space, creating destination directories, and making sure pre-requisite software is installed.

One of the reasons people really like scripting in UNIX is that the commands are the same ones you can use when typing on a shell in interactive mode, but with the added benefit of adding in "IF" and "FOR" statements.

The **sh** shell in USS is somewhat limited, but you can use the more full-featured **bash** shell later on if you would like to try something more sophisticated.

6 MAKE THE SCRIPT RUN

Go back over to your terminal session (*ssh*) and run `ls -l animals.sh`.

A terminal window titled '1: ssh' showing a series of commands and their outputs. The user is at the prompt '/z/z99994 >'. They run 'ls -l animals.sh', which shows a file with permissions '-rw-r--r--', owner '1 Z99994', group 'IPGROUP', and timestamp '2073 Aug 30 17:34'. Then they run 'chmod +x animals.sh', which changes the permissions to '-rwxr-xr-x'. Finally, they run 'cat animals.sh', which starts displaying the contents of the file.

```
TERMINAL  ...  1: ssh  +
/z/z99994 > ls -l animals.sh
-rw-r--r--  1 Z99994  IPGROUP      2073 Aug 30 17:34 ani
/z/z99994 > chmod +x animals.sh
/z/z99994 > ls -l animals.sh
-rwxr-xr-x  1 Z99994  IPGROUP      2073 Aug 30 17:34 ani
/z/z99994 > cat animals.sh
```

You should see the file shows `_r_read` and `_w_write` permissions, but no `e_xecution` permission. Next, make the file executable, by issuing the `_ch_ange _mod_e` command: `chmod +x animals.sh`

Run `ls -l animals.sh` again, and see if you can see the difference.

There's an **x** added to the permissions, letting you know that it is executable, meaning it can be run like a program.

R stands for Read, W stands for Write and X stands for Execute.

One more quick command: the `cat` command (short for `con_cat_enate`) outputs the contents of a file, to the screen. Use this to peek into the contents of a file that you don't need to edit.

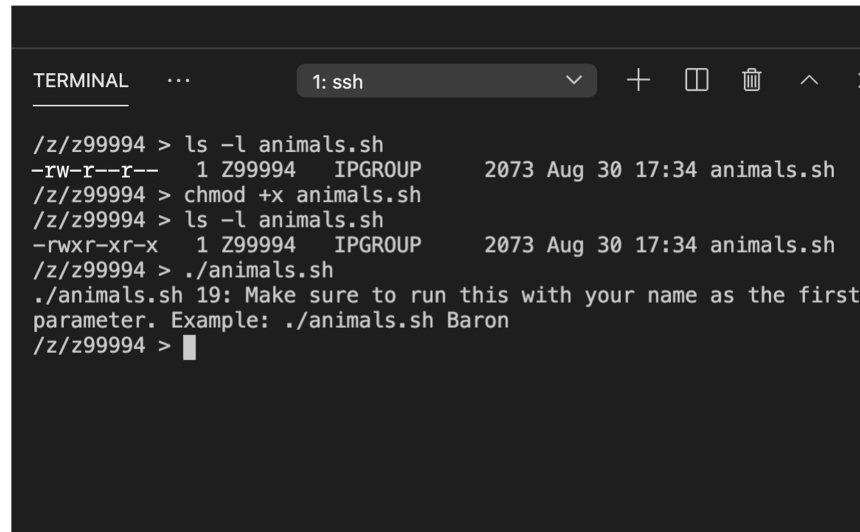
Example: `cat animals.sh`

(*more* displays files too, but shows one screenful at a time)

7 RUN THE SCRIPT

Type `./animals.sh`

In UNIX, when you give just the path to a file, it is assumed you want to run, or execute, it. (Remember to use `cat` if you want to display it, not run it)



```
TERMINAL  ...  1: ssh  +  [ ]  [X]  ^  x

/z/z99994 > ls -l animals.sh
-rw-r--r--  1 Z99994  IPGROUP    2073 Aug 30 17:34 animals.sh
/z/z99994 > chmod +x animals.sh
/z/z99994 > ls -l animals.sh
-rwxr-xr-x  1 Z99994  IPGROUP    2073 Aug 30 17:34 animals.sh
/z/z99994 > ./animals.sh
./animals.sh 19: Make sure to run this with your name as the first
parameter. Example: ./animals.sh Baron
/z/z99994 > █
```

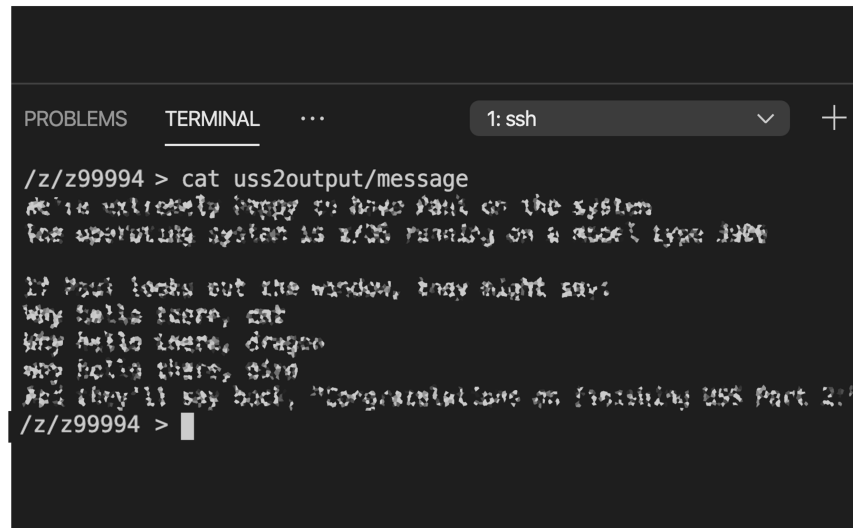
Instead of typing out the full path to the script, you can just use a single dot and a slash to say “From this current directory”, then the script file you want to run.

The script will run, but it will complain about a missing first parameter. Follow its example with your name and try again.

8 VERIFY COMPLETION

Run the *animals.sh* script again, this time giving it one argument (the part that comes after the script itself).

The script itself says what this should be, along with everything else you need to make this run correctly.



```
PROBLEMS  TERMINAL  ...  1: ssh  v  +  
  
/z/z99994 > cat uss2output/message  
We're extremely happy to have Paul on the system  
The operating system is z/OS running on a Model Type J906  
  
If Paul looks out the window, they might say:  
Why hello there, cat  
Why hello there, dragon  
Why hello there, bird  
And they'll say back, "Congratulations on finishing USS Part 2!"  
/z/z99994 > 
```

When you have finished, check the output in `~/uss2output/message` and you should see a message very clearly congratulating you on completing your task.

Look good? Feeling good? Good.

Find the **CHKUSS2** job in **ZXP.PUBLIC.JCL**, right-click on it, and then select "Submit Job" to mark this challenge complete.

Nice job - let's recap	Next up ...
<p>You took a script, figured out what it would do, and made it run. Working in UNIX System Services means you'll likely be doing this type of thing a lot, so it is good you were able to figure it out.</p> <p>We made the script full of examples on purpose, since you may want to borrow some of those expressions and conditional checks later on for your own creations, or contest submissions. (hint, hint!)</p>	<p>If you have already done the PDS and JCL2 challenges, your next task is to go after those ZOAU and Ansible challenges. These use everything you've learned so far, and puts it together in a unique way that really lets you show off your skills.</p>