Linux Commands and Shell Scripting - Final Project

Welcome to the hands-on lab for the final project!

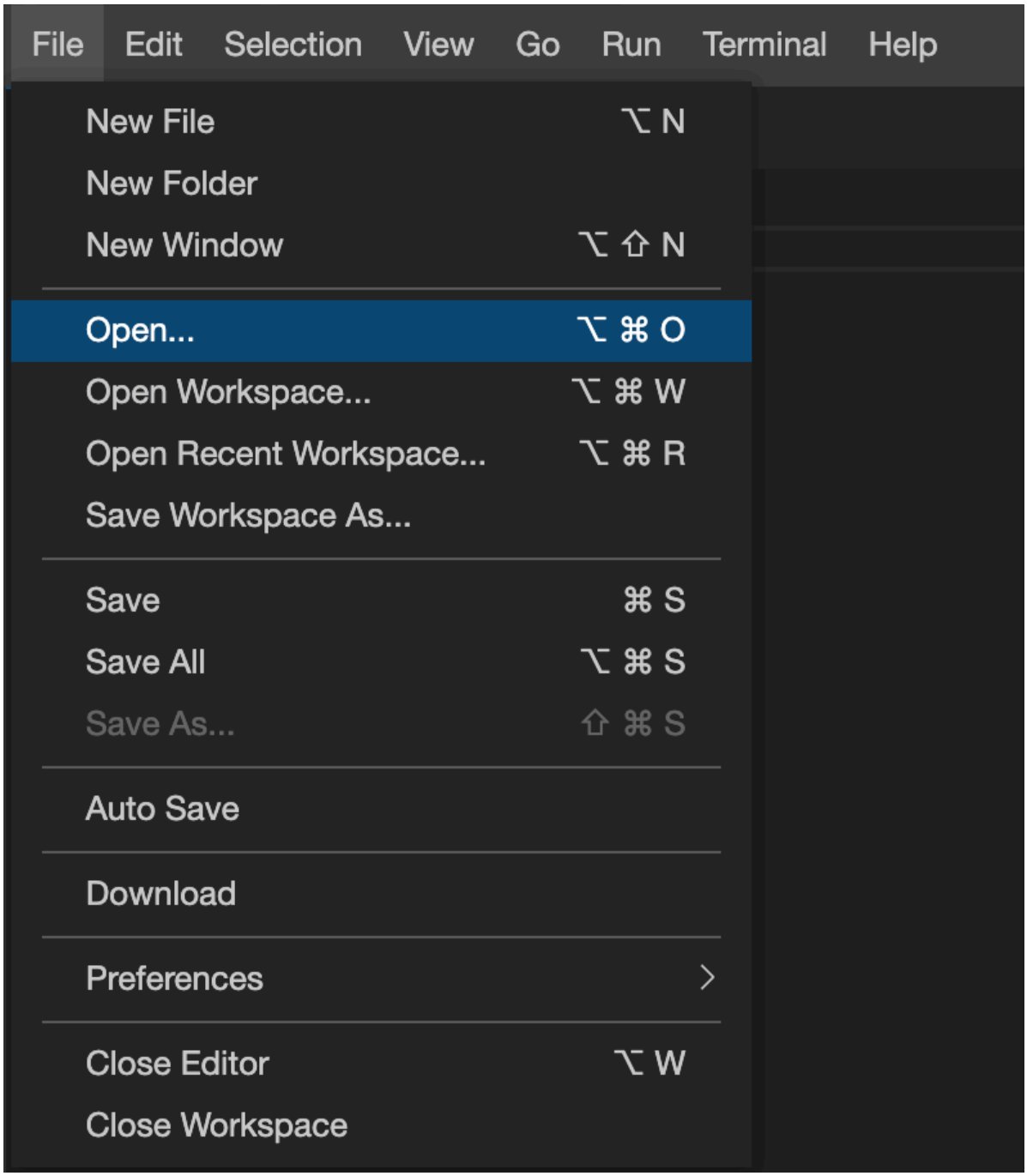
In this scenario, you are a lead Linux developer at the top-tech company ABC International Inc. As one of ABC Inc.'s most trusted Linux developers, you have been tasked with creating a script called `backup.sh` which runs every day and automatically backs up any encrypted password files that have been updated in the past 24 hours.

Please complete the following tasks, and be sure to follow the directions as you go. Don't forget to save your work.

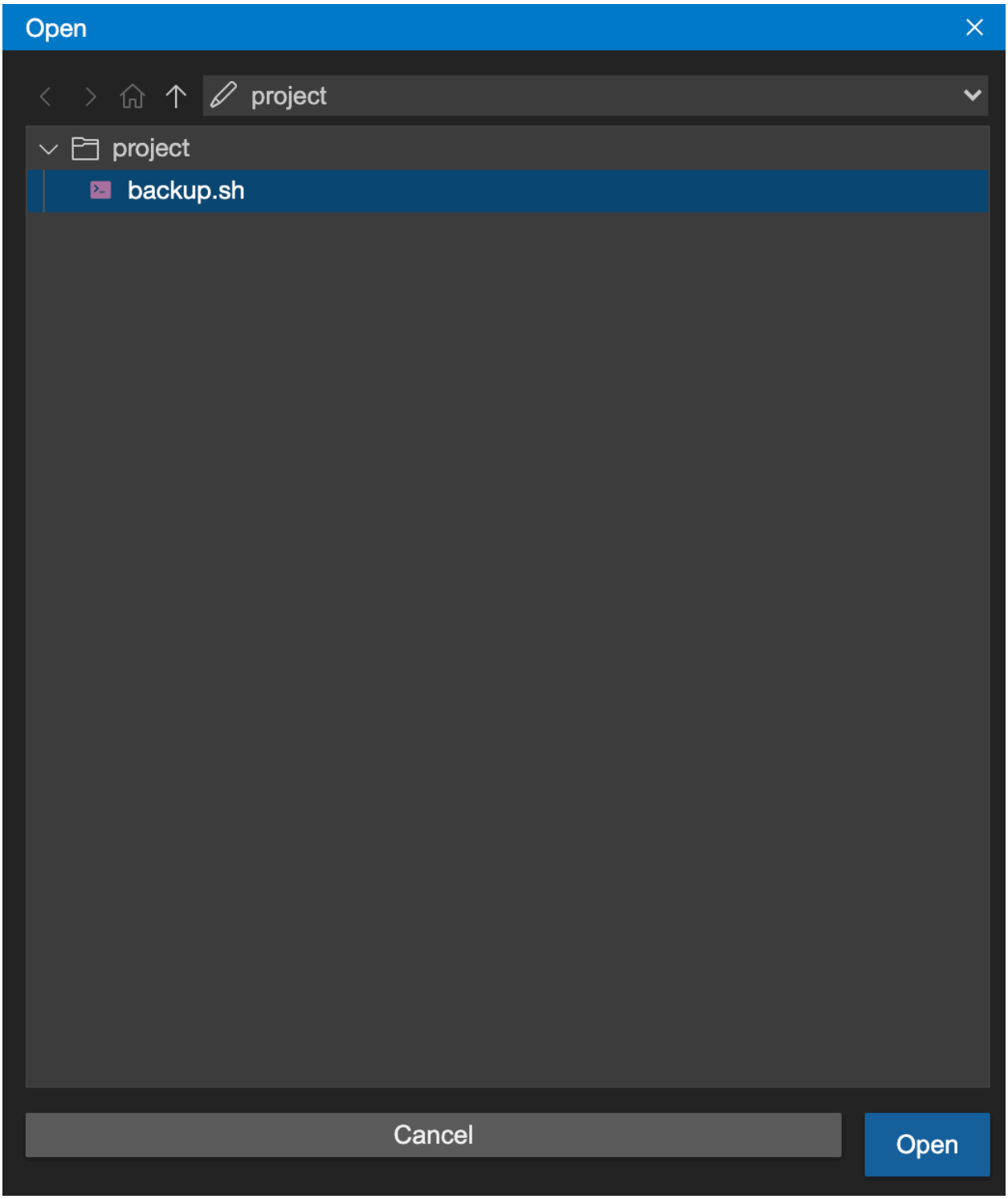
Getting started

Task 0

1. Open a new terminal by clicking on the menu bar and selecting **Terminal**>**New Terminal**:
2. Download the template file `backup.sh` by running the command below:
3. `wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/pWN3kO2yWEuKMvYJdcLPQg/backup.sh`
4. Open the file in the IDE by clicking **File**>**Open** as seen below:



then click on the file, which should have been downloaded to your `project` directory:



About the template script `backup.sh`

1. You will notice the template script contains comments (lines starting with the `#` symbol). Do **not** delete these.

The ones that look like `# [TASK {number}]` will be used by your grader:

```
backup.sh × temp.sh
backup.sh
1   # This checks if the number of arguments is correct
2   # If the number of arguments is incorrect ( $# != 2) print error message and exit
3   if [[ $# != 2 ]]
4   then
5     echo "backup.sh target_directory_name destination_directory_name"
6     exit
7   fi
8
9   # This checks if argument 1 and argument 2 are valid directory paths
10  if [[ ! -d $1 ]] || [[ ! -d $2 ]]
11  then
12    echo "Invalid directory path provided"
13    exit
14  fi
15
16  # [TASK 1]
17  targetDirectory=
18  destinationDirectory=
19
20  # [TASK 2]
21  echo ""
22  echo ""
23
24  # [TASK 3]
25  currentTS=``
26
27  # [TASK 4]
28  backupFileName=""
```

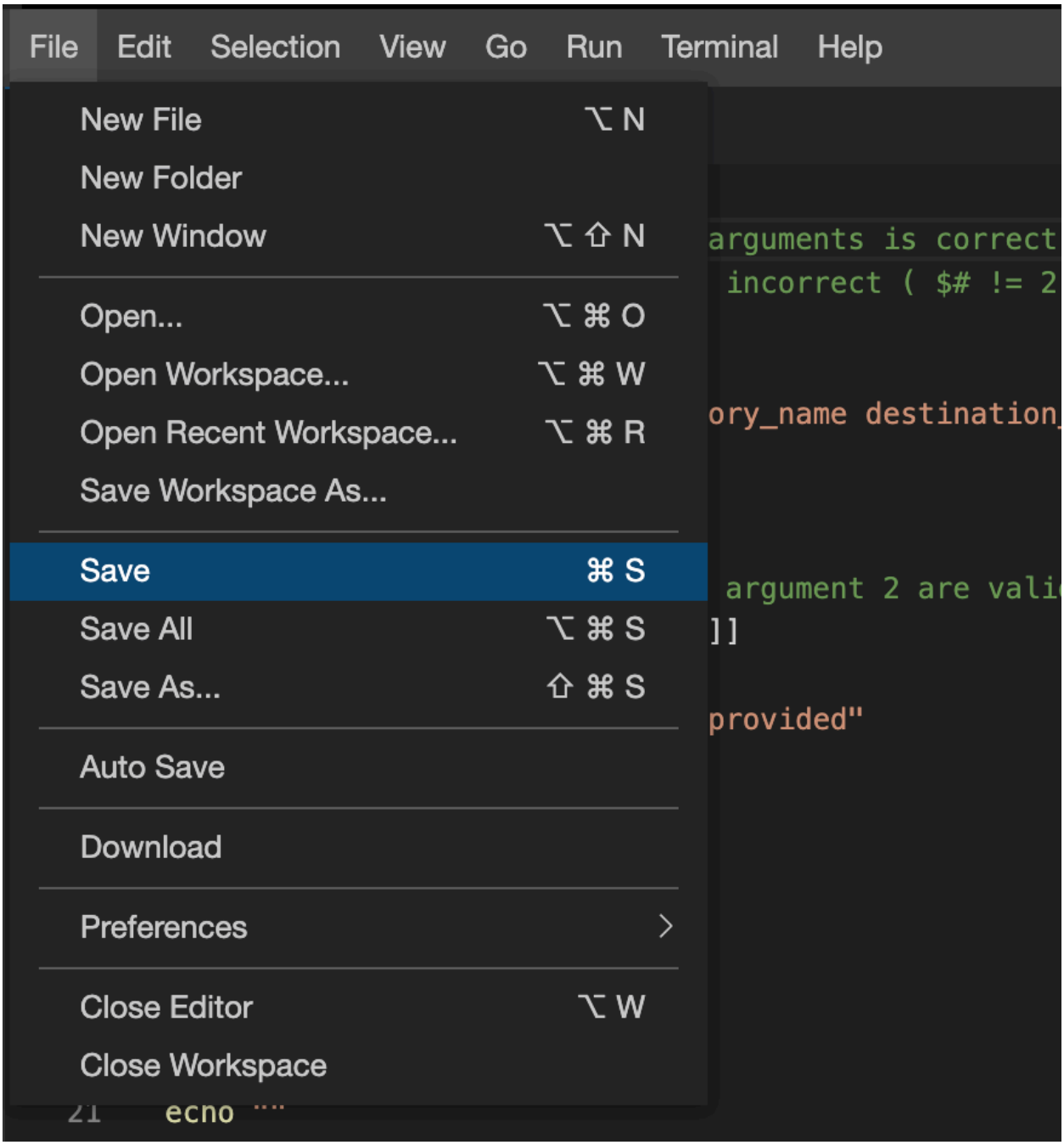
1. Also, please do **not** modify any existing code above `# [TASK 1]` in the script.

Saving your progress

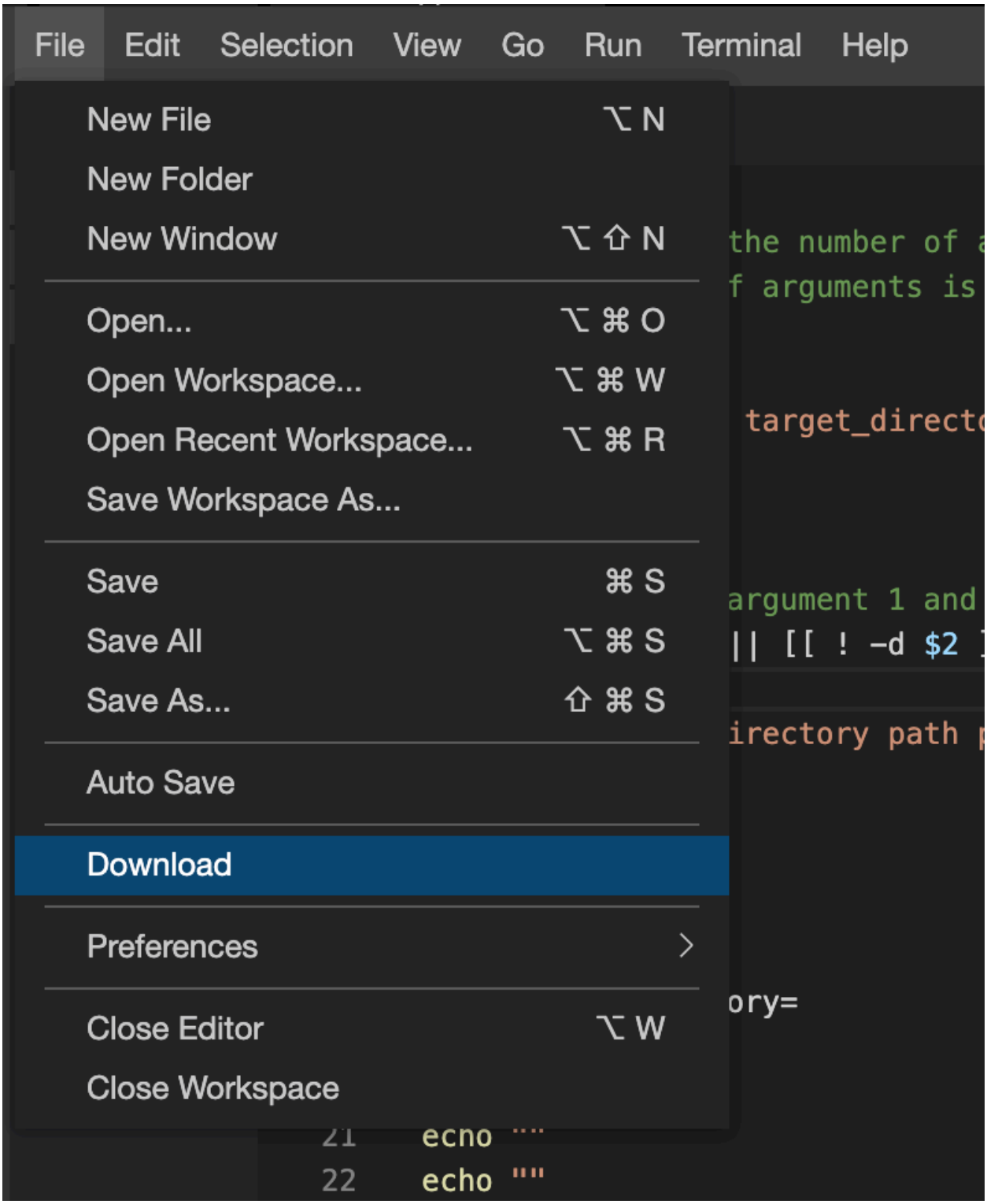
Your work **will not** be saved if you exit your session.

In order to save your progress:

1. Save the current working file (`backup.sh`)
   with `CTRL` + `s` [Windows/Linux], `CMD` + `s` [MAC], or navigate to **File**>**Save** as seen below:



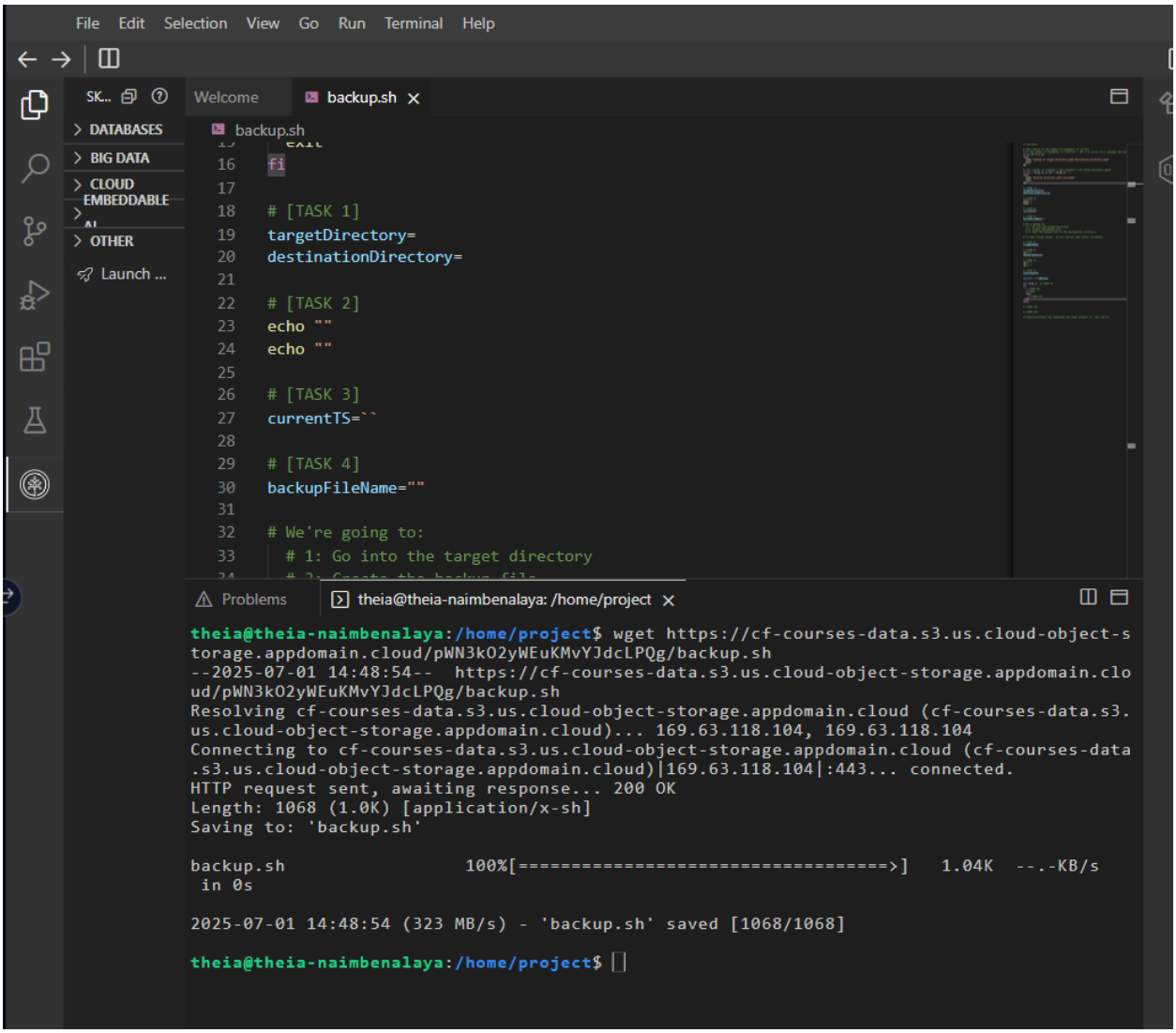
1. Download the file to your local computer by navigating to **File**>**Download** as seen below:



1. Unfortunately, our editor does **not** currently support file uploading, so you will need to copy and paste your work as follows:

- To "upload" your in-progress `backup.sh` file and continue working on it:
  1. Open a terminal and type `touch backup.sh`
  2. Open the empty `backup.sh` file in the editor
  3. Copy-paste the contents of your locally-saved `backup.sh` file into the empty `backup.sh` file in the editor

Once you run the command in the top “wget”, you will get this file



Script

```
#!/bin/bash

# This checks if the number of arguments is correct
# If the number of arguments is incorrect ( $# != 2) print error message and exit
if [[ $# != 2 ]]
then
  echo "backup.sh target_directory_name destination_directory_name"
  exit
fi

# This checks if argument 1 and argument 2 are valid directory paths
if [[ ! -d $1 ]] || [[ ! -d $2 ]]
then
  echo "Invalid directory path provided"
  exit
fi

# [TASK 1]
targetDirectory=
destinationDirectory=

# [TASK 2]
echo ""
echo ""

# [TASK 3]
currentTS=``

# [TASK 4]
backupFileName=""

# We're going to:
  # 1: Go into the target directory
  # 2: Create the backup file
  # 3: Move the backup file to the destination directory
```

```
# To make things easier, we will define some useful variables...

# [TASK 5]
origAbsPath=``

# [TASK 6]
cd # ←
destDirAbsPath=``

# [TASK 7]
cd # ←
cd # ←

# [TASK 8]
yesterdayTS=

declare -a toBackup

for file in  # [TASK 9]
do
  # [TASK 10]
  if (())
  then
    # [TASK 11]
  fi
done

# [TASK 12]

# [TASK 13]

# Congratulations! You completed the final project for this course!
```

Task 1

Navigate to `# [TASK 1]` in the code.

Set two variables equal to the values of the first and second command line arguments, as follows:

1. Set `targetDirectory` to the first command line argument
2. Set `destinationDirectory` to the second command line argument

This task is meant to help with code readability.

▼ Click here for Hint

> *The command line arguments interpreted by the script can be accessed via `$1` (first argument) and `$2` (second argument).*

Take a screenshot of the code above and save it as `01-Set_Variables.jpg` or `.png`.

```
# [TASK 1]
targetDirectory=$1
destinationDirectory=$2
```

Task 2

1. Display the values of the two command line arguments in the terminal.

▼ Click here for Hint

> *Remember, you can use the command `echo` as a print command.*
>
> - *Example:* `echo "The year is $year"`

2. Take a screenshot of the code above and save it as `02-Display_Values.jpg` or `.png`.

```
# [TASK 2]
echo "targetDirectory is $1"
echo "destinationDirectory is $2"
```

Task 3

1. Define a variable called `currentTS` as the current timestamp, expressed in seconds.

   ▼ Click here for Hint

   > *Remember you can customize the output format of the* `date` *command.*
   >
   > *To set a variable equal to the output of a command you can use command substitution:* `$()` *or* `` ``
   >
   > - *For example:* `currentYear=$(date +%Y)`

2. Take a screenshot of the code above and save it as `03-CurrentTS.jpg` or `.png`.

```
# [TASK 3]
currentTS=$(date +%s)
```

Task 4

1. Define a variable called `backupFileName` to store the name of the archived and compressed backup file that the script will create.

   > *The variable* `backupFileName` *should have the value* `"backup-[$currentTS].tar.gz"`
   >
   > - *For example, if* `currentTS` *has the value* `1634571345`*, then* `backupFileName` *should have the value* `backup-1634571345.tar.gz`.

2. Take a screenshot of the code above and save it as `04-Set_Value.jpg` or `.png`.

```
# [TASK 4]
backupFileName="backup-$currentTS.tar.gz"
```

Task 5

1. Define a variable called `origAbsPath` with the absolute path of the current directory as the variable's value.

   ▼ Click here for Hint

   > *You can get the absolute path of the current directory using the `pwd` command.*

2. Take a screenshot of the code above and save it as `05-Define_Variable.jpg` or `.png`.

```
# [TASK 5]
origAbsPath=$(pwd)
```

Task 6

1. Define a variable called `destAbsPath` whose value equals the absolute path of the destination directory.

   ▼ Click here for Hint

   > *First use `cd` to go to `destinationDirectory`, then use the same method you used in **Task 5**.*

> *Note: Please Note that you can also use the cd "destinationDirectory" || exit which ensures that if the specified directory is incorrect or inaccessible, the script will terminate immediately at this step. This acts as an implicit validation check to confirm that the correct directory is provided before proceeding with further operations. Follow the same for Task 7 .*

2. Take a screenshot of the code above and save it as `06-Define_Variable.jpg` or `.png`.

```
# [TASK 6]
cd $destinationDirectory
destAbsPath=$destinationDirectory
```

Task 7

1. Change directories from the current working directory to the target directory `targetDirectory`.

   ▼ Click here for Hint

   > `cd` *into the original directory* `origAbsPath` *and then* `cd` *into* `targetDirectory`.

2. Take a screenshot of the code above and save it as `07-Change_Directory.jpg` or `.png`.

```
# [TASK 7]
cd $origAbsPath
cd $targetDirectory
```

Task 8

You need to find files that have been updated within the past 24 hours. This means you need to find all files whose last-modified date was 24 hours ago or less.

To do make this easier:

1. Define a numerical variable called `yesterdayTS` as the timestamp (in seconds) 24 hours prior to the current timestamp, `currentTS`.

   ▼ Click here for Hint

   > *Math can be done using* `$(())`, *for example:*
   >
   > - `zero=$((3 * 5 - 6 - 9))`
   >
   > *Thus, to get the timestamp in seconds of 24 hours in the future, you would use:*
   >
   > - `tomorrowTS=$(($currentTS + 24 * 60 * 60))`

2. Take a screenshot of the code above and save it as `08-YesterdayTS.jpg` or `.png`.

```
# [TASK 8]
yesterdayTS=$(($currentTS - 24 * 60 * 60))
```

Note on arrays

In the script, you will notice the line:

```
declare -a toBackup
```

This line declares a variable called `toBackup`, which is an **array**. An array contains a list of values, and you can append items to arrays using the following syntax:

```
myArray+=($myVariable)
```

When you print or `echo` an array, you will see its string representation, which is simply all of its values separated by spaces:

```
$ declare -a myArray
$ myArray+=("Linux")
$ myArray+=("is")
$ myArray+=("cool!")
$ echo ${myArray[@]}
Linux is cool!
```

This will be useful later in the script where you will pass the array `$toBackup`, consisting of the names of all files that need to be backed up, to the `tar` command. This will archive all files at once!

Task 9

1. In the for loop, use the wildcard to iterate over all files and directories in the current folder.

   ▼ Click here for Hint

   > *The asterisk* `*` *is a wildcard that matches every file and directory in the present working directory.*

2. Take a screenshot of the code above and save it as `09-List_AllFilesandDirectoriess.jpg` or `.png`.

```
for file in $(ls) # [TASK 9]
do
```

Task 10

1. Inside the `for` loop, you want to check whether the `$file` was modified within the last 24 hours.

   > *To get the last-modified date of a file in seconds, use* `date -r $file +%s` *then compare the value to* `yesterdayTS`.
   >
   > `if [[ $file_last_modified_date -gt $yesterdayTS ]]` *then the file was updated within the last 24 hours!*

2. Since much of this wasn't covered in the course, for this task you may copy the code below and paste it into the double square brackets `[[]]`:

```
`date -r $file +%s` -gt $yesterdayTS
```

3. Take a screenshot of the code above and save it as `10-IF_Statement.jpg` or `.png`.

```
# [TASK 10]
if ((`date -r $file +%s` > $yesterdayTS))
then
```

Task 11

1. In the `if-then` statement, add the `$file` that was updated in the past 24-hours to the `toBackup` array.
2. Since much of this wasn't covered in the course, you may copy the code below and place after the `then` statement for this task:

```
toBackup+=($file)
```

3. Take a screenshot of the code above and save it as `11-Add_File.jpg` or `.png`.

```
    # [TASK 11]
    toBackup+=($file)
  fi
done
```

Task 12

1. After the `for` loop, **compress** and **archive** the files, using the `$toBackup` array of filenames, to a file with the name `backupFileName`.

   ▼ Click here for Hint

   > *Use* `tar -czvf $backupFileName ${toBackup[@]}`.

2. Take a screenshot of the code above and save it as `12-Create_Backup.jpg` or `.png`.

```
# [TASK 12]
tar -czvf $backupFileName ${toBackup[@]}
```

Task 13

Now the file `$backupFileName` is created in the current working directory.

▼ Click here for Hint

> *Move the file* `backupFileName` *to the destination directory located at* `destAbsPath` .

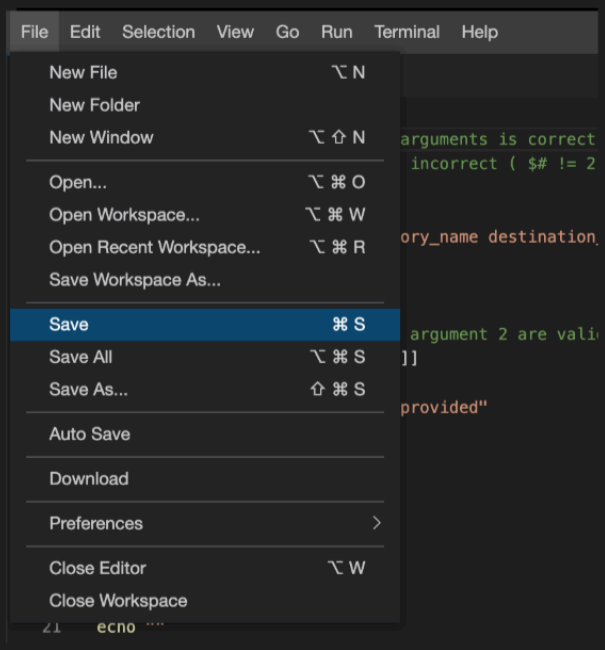
2. Take a screenshot of the code above and save it as `13-Move_Backup.jpg` or `.png` .

Congratulations! You have now done the coding portion of the lab!

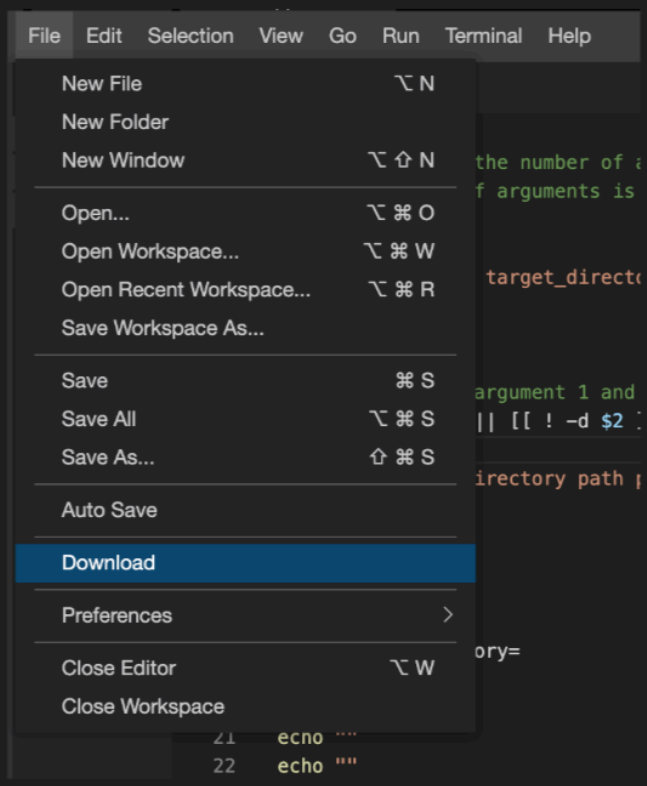
```
# [TASK 13]
mv $backupFileName $destAbsPath
```

Task 14

1. Save the current working file `backup.sh` with `CTRL` + `s` [Windows/Linux], `CMD` + `s` [MAC] or by navigating to **File->Save** as seen below:



2. Download the file to your local computer by navigating to **File->Download** as seen below:



You may save the file as `backup.sh`

3. You will later submit this file will for peer-grading.

Task 14: Upload your completed "backup.sh" file.

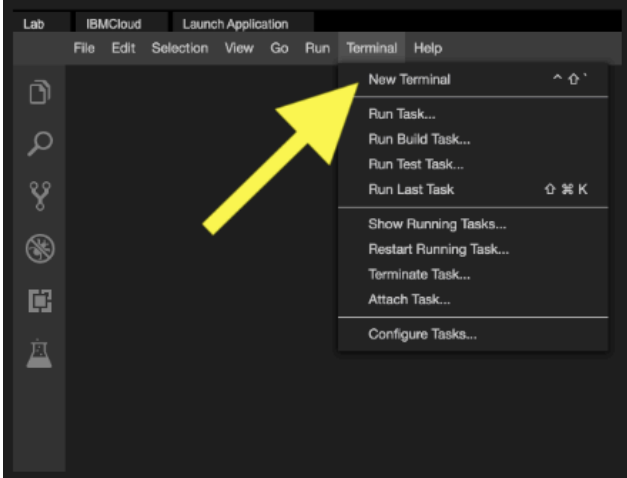
Your code will be checked and verified that all tasks are complete.

backup.sh

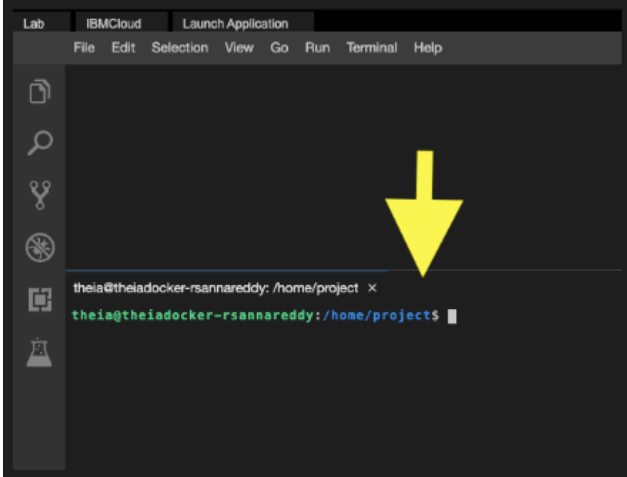
```
/home/project$ chmod u+w backup.sh
/home/project$ ls -l backup.sh
```

Task 15

1. Open a new terminal by clicking on the menu bar and selecting **Terminal->New Terminal**, as in the image below:



This will open a new terminal at the bottom of the screen as seen below:



2. Save the `backup.sh` file you're working on and make it executable.

   ▼ Click here for Hint

   > *Use the `chmod` command with the correct options.*

3. Verify the file is executable using the `ls` command with the `-l` option:

```
ls -l backup.sh
```

4. Take a screenshot of the output of the command above and save it as `15-executable.jpg` or `.png`.

```
theia@theiadocker-duongnhattha:/home/project$ chmod +x backup.sh
theia@theiadocker-duongnhattha:/home/project$ ls -l backup.sh
-rwxr-xr-x 1 theia users 1373 Sep 11 00:45 backup.sh
```

Task 16

1. Download the following `.zip` file with the `wget` command:

```
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-LX0117EN-SkillsNetwork/la
```

2. Unzip the archive file:

```
unzip -DDo important-documents.zip
```

> **Note**: `-DDo` *overwrites without restoring original modified date.*

3. Update the file's last-modified date to **now**:

```
touch important-documents/*
```

4. Test your script using the following command:

```
./backup.sh important-documents .
```

> *This should have created a file called* `backup-[CURRENT_TIMESTAMP].tar.gz` *in your current directory.*

5. Take a screenshot of the output of `ls -l` and save it as `16-backup-complete.jpg` or `.png`.

```
theia@theiadocker-duongnhattha:/home/project$ ls -l
total 16
-rwxr-xr-x 1 theia users 1373 Sep 11 00:45 backup.sh
drwxr-sr-x 2 theia users 4096 Sep 11 00:47 important-documents
-rw-r--r-- 1 theia users 4995 Sep 28  2022 important-documents.zip
```

Task 17

1. **Copy** the `backup.sh` script into the `/usr/local/bin/` directory. (Do ***not*** use `mv`.)

> ***Note***: *You may need to use* `sudo cp` *in order to create a file in* `/usr/local/bin/`.

2. Test the cronjob to see if the backup script is getting triggered by scheduling it for every 1 minute.

   ▶ Click here for Hint

3. Please note that since the Theia Lab is a virtual environment, we need to explicitly start the cron service using the below command:

```
sudo service cron start
```

4. Once the cron service is started, check in the directory `/home/project` to see if the `.tar` files are being created.
5. If they are, then stop the cron service using the below command, otherwise it will continue to create `.tar` files every minute:

```
sudo service cron stop
```

6. Using crontab, schedule your `/usr/local/bin/backup.sh` script to backup the `important-documents` folder every 24 hours to the directory `/home/project`.
7. Take a screenshot of the output of `crontab -l` and save as `17-crontab.jpg` or `.png`.

> ***Tip***: *When you are setting up cron jobs in a real-life scenario, ensure the cron service is running, or start the cron service if needed.*

```
theia@theiadocker-duongnhattha:/home/project$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
0 2 * * * /usr/local/bin/backup.sh /home/project/important-documents /home/project
```

Full Solution

```
#!/bin/bash

# This checks if the number of arguments is correct
# If the number of arguments is incorrect ( $# != 2) print error message and exit
if [[ $# != 2 ]]
then
  echo "backup.sh target_directory_name destination_directory_name"
  exit
fi

# This checks if argument 1 and argument 2 are valid directory paths
if [[ ! -d $1 ]] || [[ ! -d $2 ]]
then
  echo "Invalid directory path provided"
  exit
fi

# [TASK 1]
targetDirectory=$1
destinationDirectory=$2

# [TASK 2]
echo "targetDirectory is $1"
echo "destinationDirectory is $2"

# [TASK 3]
currentTS=$(date +%s)

# [TASK 4]
backupFileName="backup-$currentTS.tar.gz"

# We're going to:
  # 1: Go into the target directory
  # 2: Create the backup file
  # 3: Move the backup file to the destination directory
```

```
# To make things easier, we will define some useful variables...

# [TASK 5]
origAbsPath=$(pwd)

# [TASK 6]
cd $destinationDirectory
destAbsPath=$destinationDirectory

# [TASK 7]
cd $origAbsPath
cd $targetDirectory

# [TASK 8]
yesterdayTS=$(($currentTS - 24 * 60 * 60))

declare -a toBackup

for file in $(ls) # [TASK 9]
do
  # [TASK 10]
  if ((`date -r $file +%s` > $yesterdayTS))
  then
    # [TASK 11]
    toBackup+=($file)
  fi
done

# [TASK 12]
tar -czvf $backupFileName ${toBackup[@]}

# [TASK 13]
mv $backupFileName $destAbsPath
```

```
# Congratulations! You completed the final project for this course!
```