The first two assignments in PES are to be written using “vanilla” C. To minimize confusion for these assignments, I want everyone to be using the same development environment. To that end, I’ve installed a shared development server running in the cloud, which I call “CUPES” (pronounced “kyoops”). Every student has an account on CUPES.

This document explains:

* How to log in to CUPES and copy files to and from using SSH
* How to edit files on CUPES
* How to compile a simple C program on CUPES
* Technical details on the CUPES environment

# How to log in and copy files using SSH

|  |  |
| --- | --- |
| Server name | cupes.inovonics.com |
| Your username | Same as your CU Identikey; for instance, mine is “lapa3832” |
| Your identity file | You cannot log into CUPES using a password; instead, you must use public/private key login. A private/public key pair was created for you. The public key is already installed for you on CUPES. The private key was emailed to you by the course staff. Reach out on Slack if you didn’t receive it. |
| Logging in to CUPES | From the terminal on your local machine:  **ssh -i <private\_key\_file> <user>@cupes.inovonics.com**  **ssh cupes** (if you’ve set up the SSH config file as shown below)  You should see a Linux command prompt on CUPES. |
| Copying to CUPES | From the terminal on your local machine, this command will copy one or more files from your local current working directory to your home directory on CUPES:  **scp -i <private\_key\_file> <source\_file(s)> <user>@cupes.inovonics.com:**  **scp <source\_files> cupes:** (if you’ve set up the config file as shown below) |
| Copying from CUPES | From the terminal on your local machine, this command will copy one or more files from your home directory on CUPES to your local current working directory:  **scp -i <private\_key\_file> <user>@cupes.inovonics.com:<source\_file(s)> .**  **scp cupes:<source\_files> .** (if you’ve set up the config file as shown below) |
| Config file | To make all this a little easier, add these lines to your local SSH config file (usually ~/.ssh/config):  **Host cupes**  **Hostname cupes.inovonics.com**  **IdentityFile <private\_key\_file>**  **User <your\_username>**  Once you’ve done this, you can use the simpler forms of the commands above. |

The first time you try to log into CUPES, you will probably see output similar to the following:

The authenticity of host 'cupes.inovonics.com (35.206.90.220)' can't be established.

ECDSA key fingerprint is SHA256:fwwP6Oa0pzLup1ZKE+bIlKXyXQAIRRehNJDFRSr7/80.

Are you sure you want to continue connecting (yes/no/[fingerprint])?

Your ssh client is printing this message to warn you that it cannot authenticate that the machine on the other end is the same one you are trying to contact. If you were particularly worried about security, you might take some steps to verify the other side’s identity. **For now, you should just type “yes”.** Your ssh client will store the key from CUPES so that in the future it can verify that machine’s identity.

You might also see an error message similar to this:

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ WARNING: UNPROTECTED PRIVATE KEY FILE! @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Permissions 0666 for 'muda7613' are too open.

It is required that your private key files are NOT accessible by others.

This private key will be ignored.

Load key "muda7613": bad permissions

If you get this message, you will need to restrict the permissions for your private key file. To do so, type the following command: **chmod 600 <your-key-file>**

If you are unfamiliar with the Linux command line, there is a reasonable tutorial at <https://ubuntu.com/tutorials/command-line-for-beginners>. Also, I encourage you to use Google when you have a technical question, especially about Linux; it is exceedingly likely that your Linux question is answered somewhere on the Internet.

# How to edit files on CUPES

Broadly speaking, there are three approaches to editing files on a remote Linux server like CUPES:

1. **Edit the files directly on CUPES.** Three editors—nano, vi, and Emacs—are installed on CUPES. Of these three, nano is the easiest to use as a newcomer. Vi is only to be used if absolutely needed to prevent a zombie apocalypse. If you choose to use Emacs, meanwhile, you will signal that you are an engineer of the highest possible refinement, with a nuanced eye for superior tools.

But in all seriousness, there is a relatively large learning curve for both Emacs and vi.

To use one of these editors, log into CUPES and type (for instance) **nano <filename>** from the command line. Documentation for all three is copiously available online, so use Google as needed.

1. **Set up your local editor to access files on CUPES via SSH.** Most editing applications intended for professional software developers know the SSH protocol and can be configured to access the files on CUPES while running on your laptop. Once configured, you should be able to transparently open and save the files on CUPES. Google your editor’s documentation to figure out how to make it work.
2. **Edit the files locally, and then copy them from your machine to CUPES.** This is the most cumbersome and error-prone approach, but it’s also the one you probably know how to do right now. Rather than copy individual files (even *more* error-prone!), I would set up a local directory that is a complete mirror of the CUPES directory. Then any time you change a file locally, you will want to issue a command like **scp \* cupes:path/to/remote/directory** from your local command prompt (assuming your current working directory locally is the directory you are mirroring on CUPES). This will push the *entire contents* of the local directory to CUPES, which may slightly help prevent the problem of files getting out of sync.

# How to compile and run a simple C program on CUPES

Let’s assume you have created the file hello.c in a directory on CUPES using one of the editing approaches above. You can compile it easily from the command line:

**cd path/to/project/directory**

**gcc -Wall -Werror hello.c -o hello**

If you have any compilation warnings or errors, you will see them here; otherwise you can run your program with the command **./hello**

A debugger is not necessary for the first two assignments, but it can be a powerful way to help pinpoint programing errors. To debug your program from the command line, you’ll want to add the -g flag to gcc when compiling. You can then use gdb to debug:

**gcc -Wall -Werror -g hello.c -o hello**

**gdb ./hello**

A tutorial for gdb is available at <http://www.gdbtutorial.com/>. We will also be covering a bit of gdb basics later in the course.

# Technical details on the CUPES Environment

Information in this section is for the curious—you do not need to understand any of this to complete the first two assignments.

CUPES is running on an GCP virtual machine, specifically GCP’s e2-small instance type. The underlying processor is allocated based on availability, and CUPES is configured to have 2 virtual CPUs and 2 GB of RAM. The server is run out of one of GCP’s central1-a zones.

The operating system is Ubuntu 20.04 LTS.

On top of the GCP configuration, I am using a Azure DNS service(through Inovonics) to provide Dynamic DNS service, which explains why the full system name is cupes.inovonics.com.