https://github.com/zalewis2/ser321-spring26-C-campus-zalewis2

Assignment 1
OSI model

Version: January 8, 2026

This assignment is mostly about setup and learning things you will need in the future: working on the command line, working with Gradle and the example repo. My advise is to actually research the command line arguments and understand them and not just searching for an answer.
You will need a PDF or markup document that you will submit on Canvas AND in your GitHub repo (see below - ser321-spring26-C-campus), I will call this "document" during the assignment description. Your document needs to be formatted well, so we can navigate it easily and find your answers and can map it to the correct task. If that is not fulfilled you might lose up to 5 points in this assignment (these points are not listed anywhere, these are additional deductions).
Let's start of with practicing some command line commands and some setup.

Part I.
Linux, Setup (35 points)

Prerequisites:

1. Familiarize yourself with the command line (see lecture materials for

examples). 2. Review the relevant information posted on Canvas.

Learning outcomes for this first part:

1. Efficiently use the Linux command line for file and directory management.
2. Gain comfort with command line pipelining and I/O redirection.

This assignment will sometimes tell you the specific commands to use, but it is up to you to read the *man* page (available from Linux) in order to be successful or use the internet for your research.
Note: This assignment is designed for a Linux environment. If you use a different sys tem, clearly state which one and note any command differences. Only Linux is officially supported. **We only support Linux.** You do not have to do this on AWS but you can. In your document mention which system you are working on if you are not working on Linux, e.g. Windows Version so we know why the commands differ.

1. Command line tasks (5 points)

Deliverable

This part has very few points and it is more about going through these since I think it is important that you can handle the command line a bit. When grading we will go through this quickly and random check some of the commands to get the 5 points. Complete the following tasks using only the command line (no GUI tools). For each task, write the command you used, following the numbering below.
Linux System:
1: mkdir cli_assignment
2: Your Command 2
3: Your command 3

Tasks

You start in any directory, preferably a test folder. You must work in a command line; you are not allowed to use your FileExplorer or Finder to accomplish these tasks. Your individual answers are worth between 0.25 and 1 points, partial credit will be given as well.

1. Create a directory named "cli_assignment".

   mkdir cli_assignment

2. Change the current working directory to the new directory created in the previous step.

   cd cli_assignment

3. Create a new file named "stuff.txt". Use the *touch* command to do this. Read about the touch command using the manual (*man*) pages.

   touch stuff.txt

4. Add some text (multiple lines) to this text file using the *cat* command.

   cat > stuff.txt

   this is stuff for the stuff text file

   this is also for the file

   hello

5. Count the number of words and the number of lines in the file "stuff.txt".

wc -w < stuff.txt (for words)

result = 14

wc -l < stuff.txt (for lines)

result = 3

6. Append more text to the file "stuff.txt".

cat >> stuff.txt

7. In the current working directory, create a new directory "draft".
mkdir draft

8. Move the "stuff.txt" file to the directory "draft".

mv stuff.txt draft/

9. Change your working directory to "draft" and create a **hidden** file named "secret.txt".

cd draft

touch .secret.txt

10. Create a new directory ("final") as a copy of the "draft" directory (final should be on the same level as draft) using the copy command.

cd ..

cp -r draft final

11. Rename the "draft" directory to "draft.remove". Use the **mv** command for this.

mv draft draft.remove

12. Move the "draft.remove" directory inside the "final" directory. Use the **mv** command for this.

mv draft.remove final/

13. From inside the "cli_assignment" directory, list all the files and sub-directories and their permissions.

cli assignment (final)

final (draft.remove final stuff.txt)

draft.remove (final stuff.txt)

final (final stuff.txt)

14. List the contents of the given file "NASA_access_log_Aug95.gz" without extracting it. (The file should be on the same level as your "cli_assignment" directory)
    zless NASA_access_log_Aug95.gz

15. Extract the given file "NASA_access_log_Aug95.gz".

    gunzip NASA_access_log_Aug95.gz

16. Rename the extracted file to "logs.txt".

    mv NASA_access_log_Aug95 logs.txt

17. Move the file "logs.txt" to the "cli_assignment" directory.

    mv logs.txt cli_assignment

18. Read the top 100 lines of the file "logs.txt".

    head -n 100 logs.txt

19. Create a new file "logs_top_100.txt" containing the top 100 lines using I/O redirection.

    head -n 100 logst.txt > logs_top_100.txt

20. Read the bottom 100 lines of the file "logs.txt".

    tail -n 100 logs.txt

21. Create a new file "logs_bottom_100.txt" containing the bottom 100 lines using I/O redirection.

    tail -n 100 logst.txt > logs_top_100.txt

22. Create a new file "logs_snapshot.txt" by concatenating files "logs_top_100.txt" and "logs_bottom_100.txt".

    cat logs_top_100.txt logs_bottom_100.txt > logs_snapshot.txt

23. Now append (to the end) to the "logs_snapshot.txt" the line "asurite: This is a great assignment" and the current date (asurite is your asurite, e.g. amehlhas for me).

    echo "zalewis2: This is a great assignment 1/30/26" >> logs_snapshot.txt

24. Read the file "logs.txt" using the less command.

    less logs.txt

25. Using the given file "marks.csv" (delimited by %), print the column "student_names" without the header (you can use the column num as index). Use the cut command and I/O redirection. (This file should be on the same level as your "cli_assignment" directory).

    tail -n +2 marks.csv | cut -d'%' -f1 > student_names.txt

26. Using the given file "marks.csv", print the sorted list of marks in "subject_3". Use the sort command piped with the cut command.

    cut -d'%' -f4 mark.csv | tail -n +2 | sort -n

27. Using the given file "marks.csv", print the average marks for "subject_2" (it is ok to us awk).

    awk -F '%' 'NR>1 {sum += $4; count ++} END {print sum/count}' marks.csv

28. Save the average into a new file "done.txt" inside of the "cli_assignment"

directory.

    awk -F '%' 'NR>1 {sum += $4; count ++} END {print sum/count}' marks.csv >

cli_assignment/done.txt

29. Move "done.txt" into your "final" directory.

    mv done.txt final

30. Rename the "done.txt" file to "average.txt".

    cd final

        mv done.txt average.txt

2. Some Setup and Examples (30 points)

2.1. Setup a GitHub repo so you version control your work (5 points)

1. Create a **private** GitHub repository named "ser321-spring26-C-campus" (replace <asurite> with your ASURITE ID).
2. Invite ser316asu as a collaborator.
3. Create the following folders in your repo: Assignment1, Assignment2, Assignment3, Assignment4, Assignment5, Assignment6. (Tip: Add a placeholder

file like .gitkeep to commit empty folders.)

4. Upload your solution for this assignment to the Assignment1 folder.
5. Submit your repo link at the top of your assignment document and as a comment in your Canvas submission.

   https://github.com/zalewis2/ser321-spring26-C-campus-zalewis2

I advise you to use Git while you work on your programming assignments later on even though you will submit a zip on Canvas. For every assignment, you will always need to complete the following:

1. Upload your solution to GitHub (into the correct assignment directory).
2. Add your repo link to the submission box on Canvas.
3. Zip up your assignment X directory and submit it on Canvas as well. The zip should only contain the current assignment
4. Other necessary tasks will be specified in each individual assignment, for example, what to do with READMEs, PDFs, or screencast links.

We need to have your whole submission submitted as specified here or if mentioned differently in the assignment document you should follow that. If it is not submitted correctly it will lead to 0 points. You might be allowed to fix small things for a penalty of -10% after grading is done.

Deliverable: Add the GitHub repo link to the TOP of your assignment document (PDF) which you submit on Canvas AND as a comment in your Canvas submission. This is graded as well!

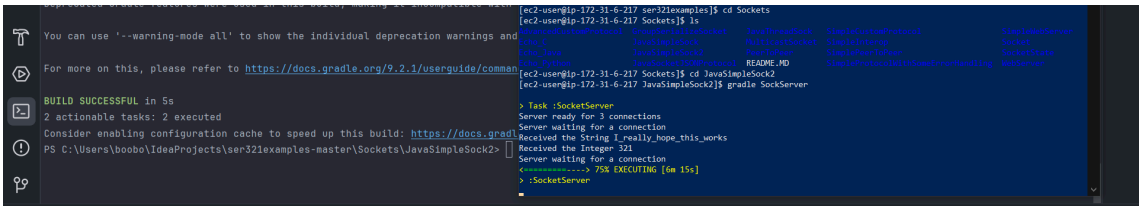
2.2. Running examples (10 points)

Go to the Example GitHub repo.

1. I advise you to fork this repo, so you can make changes and have your own version. On the Readme of the repo it explains how to fork.
2. Run three different example projects (from different directories) using Gradle on the command line.
3. For each:
   a) Name the example.
   b) Include a screenshot of your terminal showing the example running.
   c) Briefly explain what the example does.

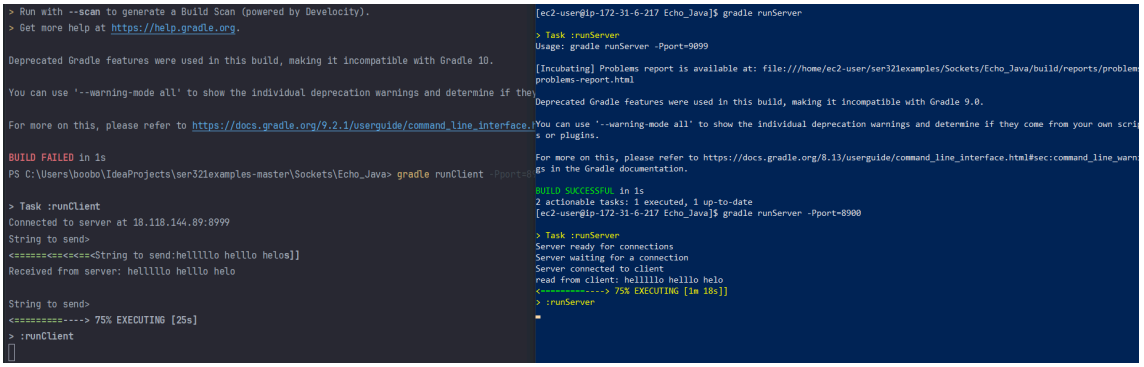
1) JavaSimpleSock2

Ran and connected SockServer.java and SockClient.java. Server opens on port 8888, and waits for Client. The server is open for 3 messages and 1 number for each message sent.



2) Echo_Java

This runs server and client. Anything that is typed in the client is echoed/ repeated over on the client side.



3)

~~Deliverable~~: Name the three examples you ran and take a screenshot for each of your examples from your command line (if you needed more than one Window show all of them in your screenshot) so we see you did run them. Add these to your document and make
sure we can actually read the screenshots. Ensure your document is well formatted and does not "waste space". Explain each example and what you think it does briefly.

3 points per example (naming the example, screenshots, explanations), 1 point for having it well formatted and readable.

2.3. Understanding Gradle (7 points)

You are not supposed to make some changes to Gradle/Java. Do the following:

1. Copy the Gradle/JavaGradle example from the example repo to your Assignment1 directory
2. In the build.gradle file in that example you will find a "Try:" with some tasks, make the appropriate changes in the build.gradle file and Java files.

~~**Deliverable:**~~ Commit your changes to your repository.

2.4. Set up your second system (8 points)

1. Set up your second system (e.g., AWS EC2, see Canvas) with JDK, Gradle, and Git.
2. Clone both your assignment repo and your forked examples repo to this machine and your local machine.
3. Run the **JavaSimpleSock2** example with the server on your second system and the client on your main system. Go to the Canvas AWS page; the last video on there explains how to run the JavaSimpleSock example on AWS (for you it it JavaSimple Sock2.
4. Record a short screencast showing the example running on both

systems.

5. Include a link to your screencast in your assignment document.

~~**Deliverable:** In your Deliverable document, add a link to your screencast showing the example running on your AWS system and your client running locally on your system.~~

Part II.
Networking (43 points)

3. Network traffic

This section comprises a collection of activities exploring networking protocols and tools for the data link, network (IP), and transport (TCP) layers of the network stack. Advice: Do not just copy and paste the command from this document; this sometimes leads to issues when done from a PDF. I would also advise you to spend some time understanding what you are doing and not just blindly following the commands.
You will need to create a document in which you will add all screenshots, answers, etc. Please structure your document according to the tasks and activities in the assignment. If you want to you can of course remove information from your screen shots if you think they are things you want to keep private.

Learning outcomes

- Understand the nature of network traffic on your local computing device.

Objectives

- Understand the nature of network traffic at the data link, network, and transport layers.
- Gain basic competency using common command-line and GUI tools to analyze net work traffic and routing.
- Identify the structures in IP and TCP packets.
- Understanding packet routing (paths) across the open Internet.

Tools needed

- Ipconfig / ifconfig provide information regarding your network interfaces.
- The watch command will run a command repeatedly on a set interval (Unix).
- The netcat command gives you a command-line network tool to communicate over sockets using specific protocols and ports.
- The netstat command displays comprehensive network information, including socket states
- Wireshark is a packet sniffing application with mature facilities to filter and capture network traffic. There is an overview video on using Wireshark in the course shell.
- Traceroute and ping may be used to show routes taken by IP packets through dif ferent routers ("hops") using ICMP messages.

3.1. Understanding TCP network sockets (11 points)

1. The network should be observed for a period of at least 10 minutes during which a decent amount of network activity takes place. Network activity in this context would mean basic web browsing - emails, social media, streaming. Ideally a mix of the mentioned activities. So basically run the command I ask you to figure out below and then generate some network traffic.
2. This network activity observation should be done via netstat.

You'll need to continuously monitor the output of the netstat command. This must be automated/scripted in command line/bash.

a) The watch, grep, and netstat commands could be one way to do this.
b) Grab readings every 30 seconds and filter for socket connections that are in state ESTABLISHED or LISTEN (this should all be done with a command line or bash script - do not filter manually). You will need to find out the command

to use here. Tip: print the current time every 30 seconds before the data that you are grabbing so you can distinguish the different time steps.

3. Once you have the required data points, import them into Excel or a suitable graph ing tool and make a line chart of each socket state count over the 10-minute period. So show how many Sockets are listening and established at every 30sec mark (two lines).

Deliverable: Include the command/script you used and the graph of socket states over the 10 minutes in your document.

3.2. Sniffing TCP/UDP traffic (15 points)

Step 1 (TCP) (7 points)

1. Setup Wireshark on your primary computer (not AWS) to intercept and log network traffic. Important note: Loopback traffic is a little different on Windows, see here. You will call both commands on your primary computer, OR alternatively another option is to call the first command on AWS, then the second on your local computer. To complete this alternative option, you will need to open the port - you can change the port - and use the AWS public IP instead of 127.0.0.1. The interface used in Wireshark would also need to be your network interface and not Loopback.
   a) Start a trace on the Loopback: lo0 interface with a filter on the tcp port, tcp.port=3333. The port here could be anything between 1K and 48K, but let's use 3333.

2. To generate traffic we are going to use netcat. netcat is a utility program for reading from and writing to network connections using TCP or UDP.
   a) To read, run on the command line: nc -k -l 3333. Keep the command window open after running this. Please make sure that port number here (3333) matches the port number set as a filter in step 1a and the port number in 2b.
   b) Open a second command window. To write, run on the command line: nc 127.0.0.1 3333. After running the command, in this window, type in:

   SER321

   – then hit enter, then type

   is amazing!!!

   – then hit enter

   c) Make sure that you type the above as it is, with a newline in between.

3. Now stop both the commands from step 2, press Ctrl + C on the terminal windows running the commands.

4. Stop the Wireshark recording and take a screenshot of your data with the appropriate filter.
5. Answer the following questions (1 points for each)
   a) Explain both the commands you used in detail. What did they actually do?

b) How many packets were send back and forth so the client/server could send/re ceive these two lines of data?

c) How many packets were needed back and forth to capture the whole "process" (starting the communication, ending the communication, sending the lines)?

d) How many bytes is the data (only the data) that was sent from client to

server? e) How many total bytes went over the wire (back and forth) for the

whole process?

f) How much overhead was there. Basically how many bytes was the whole process compared to the actually data that we did send.

Step 2 (UDP) (8 points)

Basically, do the same thing as above just with UDP, I will only give the basics here.

1. Setup Wireshark to intercept and log network traffic.
   a) Start a trace on the Loopback: lo0 interface with a filter on the tcp port, udp.port=3333. The port here could be anything between 1K and 48K.
2. To generate traffic:
   a) Run: nc -k -l -u 3333.
   b) Run: nc -u 127.0.0.1 3333. After running the command, in the same window, type

      SER321

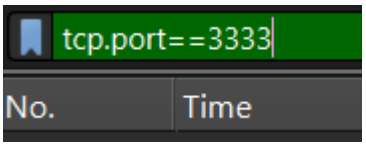
      Rocks!

   c) Make sure that you type the above as it is, with a newline in between.
3. Stop the client and server.
4. Stop the Wireshark recording and take a screenshot of your data with the appropriate filter. 1 points for each:
   a) Explain both the commands you used in detail. What did they actually do?
   b) How many packets were needed to capture those 2 lines?
   c) How many packets were needed to capture the whole "process" (starting the communication, ending the communication)?
   d) How many bytes is the data (only the data) that was sent?
   e) How many total bytes went over the wire?
   f) Basically how many bytes was the whole process compared to the actually data that we did send?
   g) What is the difference in relative overhead between UDP and TCP and why?

Specifically, what kind of information was exchanged in TCP that was not exchanged in UDP? Show the relative parts of the packet traces.

Deliverable: Put the following in your document

• TCP: Screenshot of Wireshark capture with appropriate filter (1 points) •



TCP: Answers to all the questions in Step 1 (6 points)

• UDP: Screenshot of Wireshark capture with appropriate filter (1 points) •

UDP: Answers to all the questions in Step 2 (7 points)

3.3. Running client servers in different ways (17 points)

In this section I want you to run the JavaSimpleSock2 example from the repo. You will also need to have Wireshark open to capture the traffic between client and server. This is similar to the setup you already did but here we also want to look through Wireshark.

3.3.1. Running things locally (6 points)

Run your client and server locally on your computer (so both of them on localhost). Setup Wireshark so that it shows the traffic for your client/server connection. Send a string and integer to the server using the appropriate command line arguments and wait for the response.
Take a short screencast (max. 2min) and add the link to this video to your document. In this video talk us through the Wireshark traffic and show us where the string and integer are sent to the server and where to find the response on Wireshark. In your document add screenshots of your command line windows showing all the com mands you called and the output in the console.

3.3.2. Server on AWS (5 points)

Now, run your server on AWS and your client locally on your computer. Make sure you use the correct IP and port and that the port on AWS is opened for traffic. Setup Wireshark to capture the traffic again and send over some data as before. Add screenshots of your command line windows to your document. Also describe what changed compared to when just running things locally, what did you need to change on Wireshark, what did you need to change in your Gradle calls (where there changes?)?.

3.3.3. Client on AWS (3 points)

Consider the case you want to run your server locally (so on your home computer) and your client on AWS (you do **not** have to do this but you can try). Does this work without issues? Can you do it in the same way as in 3.3.2? Why or why not? What is different?

3.3.4. Client on AWS 2 ((3 points))

In this context also explain how the differences in local IP addresses, how your router plays into all of this. Why can you easily reach your server on AWS with a client running in your local network but not as easily in the other direction (using a client on AWS or a computer on an external network and connecting to a server that is within the local network)? And what can you do to reach your server in your local network if you want to reach it from outside your network (you do not have to do that)? What is the "issue" if you want to run your server locally and reach it from the "outside world"?

Deliverable:
Answer all questions from above, non of them are rhetorical in this section. Remember the link to your screen cast, your screenshots and explanations.

4. Submission

On Canvas submit the link to your GitHub repo "ser321-spring26-C-campus". In the main/master branch of this repo you should now have 6 directories (one for each assign ment). On Canvas also submit your PDF and then separate your zip file with everything from the Assignment1 directory.
In the Assignment1 directory on GitHub you should have:

- Assign1-2.pdf (or markup) document with all your Command line commands and all your answers/explanations from the other sections into your Assignment1 folder on GitHub. Remember this needs to be well formatted (this will be graded)