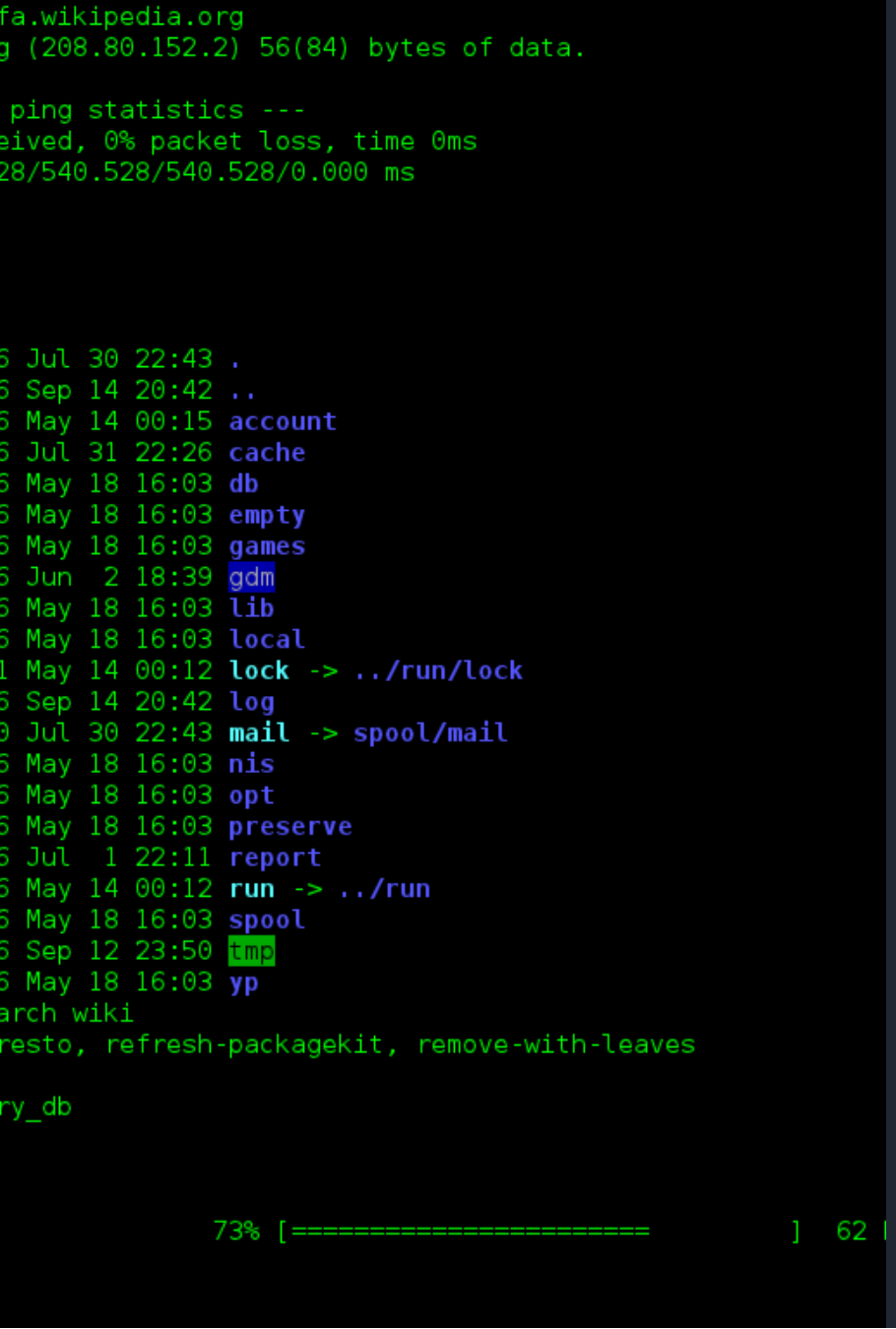
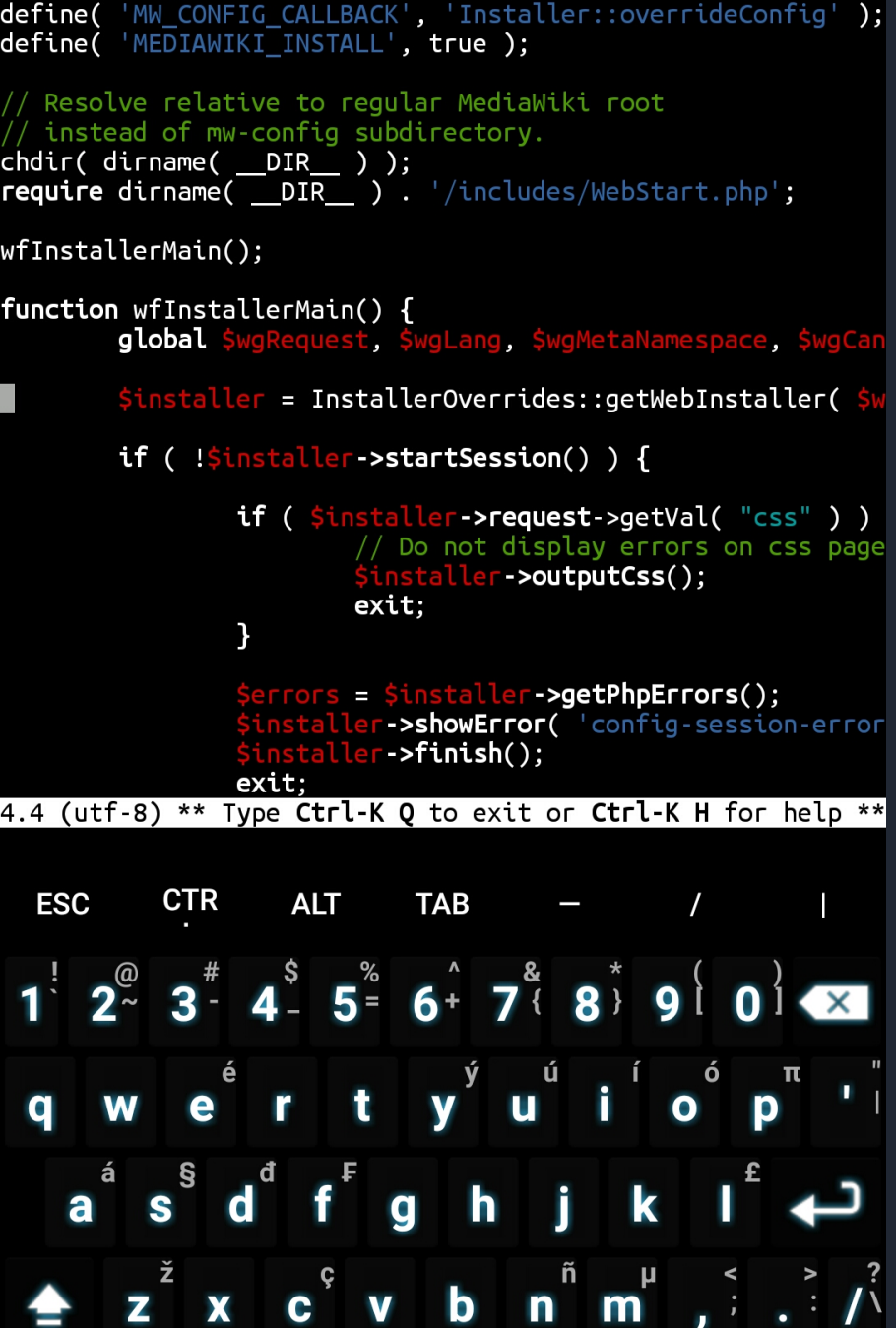


Linux Shell Scripting

Linux shell scripting is a powerful way to automate tasks, perform system administration, and create custom utilities in a Linux environment. Here are the basics:

by sahitya



Choose a Shell

Linux offers several shells, with bash being the most widely used and recommended for beginners.

Create a Shell Script File

Use a text editor like nano, vim, or gedit to create a shell script. Ensure that the file has execute permission (chmod +x script.sh).

Shebang Line

Start your script with a "shebang" line, which specifies the shell interpreter to use. For `bash`, use `#!/bin/bash`.

Comments

Use comments to explain your script's purpose, how it works, and any important information. Comments start with `#` and are ignored by the shell.

Variables

Declare and assign values to variables. Variable names are case-sensitive and can contain letters, numbers, and underscores.

User Input

Read user input using the `read` command.

Echo/Print Output

Use `echo` to display output to the terminal.

Command Execution

Execute shell commands and capture their output using backticks or `$()`.

Conditional Statements

Use `if`, `elif`, and `else` for conditional branching.

Loops

Create loops using `for` and `while`.

Functions

Define functions to encapsulate code for reuse.

File Operations

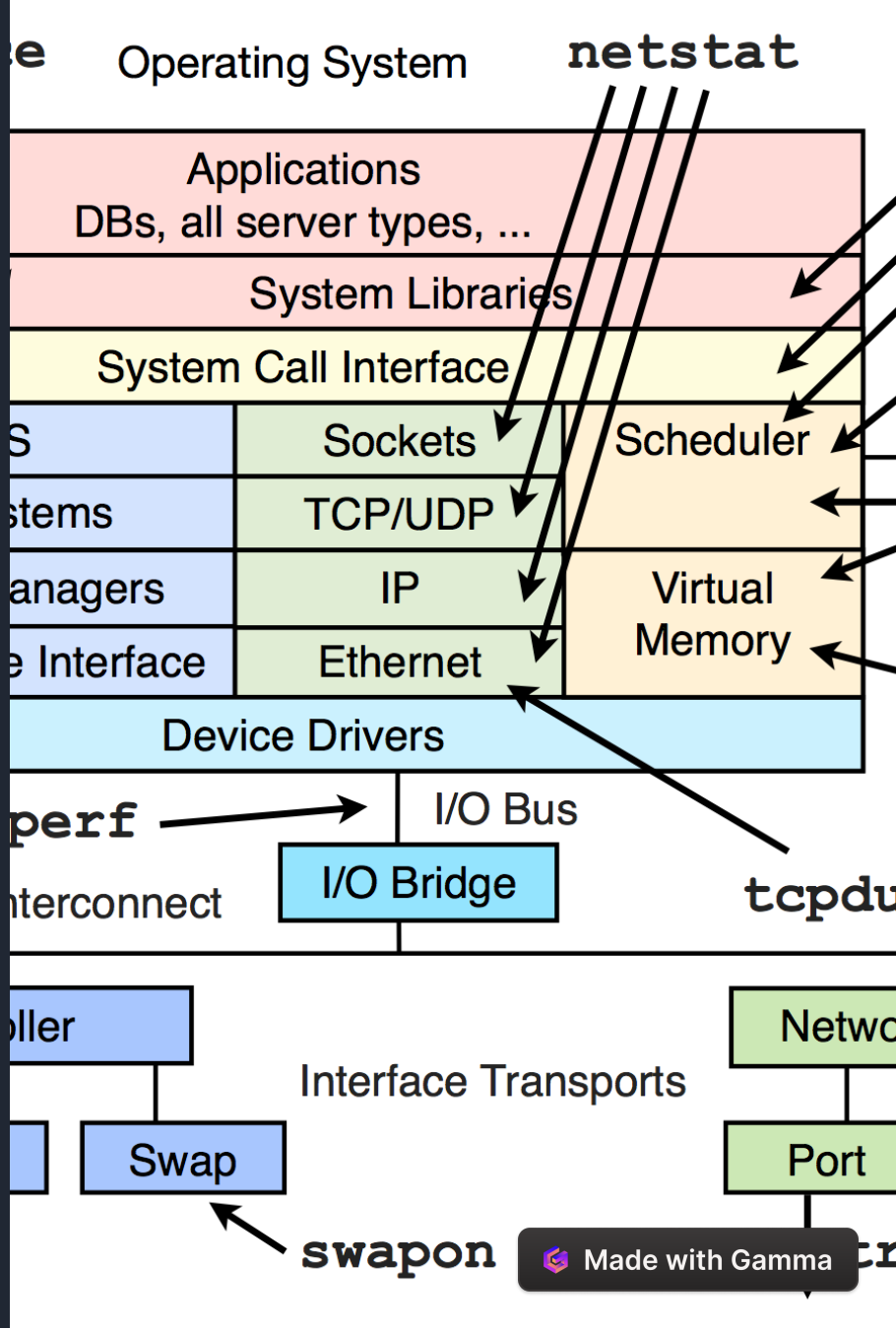
Manipulate files and directories using commands like `touch`, `mv`, `cp`, `rm`, `mkdir`

Error Handling

Use `set -e` at the beginning of your script to exit if any command returns a non-zero status. You can also use `trap` to catch and handle errors., and `cd` within your script.

Testing and Debugging

Use `echo` or `printf` statements for debugging. You can also use tools like `shellcheck` to analyze your scripts for syntax and style issues



Permissions

Ensure your script has the execute permission (`chmod +x script.sh`) to run it as a standalone program

Execution

Run your script by typing `./script.sh` in the terminal.

Example Script 1

Here's an example script that extracts lines from a specified file based on a specified range of line numbers:

```
root@DESKTOP-6872CI7:/home# cat > DisplayData.sh
echo "enter the filename"
read fname
echo "enter the start"
read s
echo "enter the ending line number"
read e
sed -n $s,$e\p $fname | cat > newline
cat newline
```

This script prompts the user to enter a filename, a starting line number, and an ending line number. It then uses the sed command to extract the specified range of lines from the specified file and displays those lines in the terminal.

```
sses are controled by a file RUN_STEPS


level set for $DEBUG_LEVEL" >> ${LOGRUN_

n_all_somatic_SNV_steps


 ${RUN_STEPS}

ute_low_somatic_snp" >${RUN_STEPS}
ute_high_somatic_snp" >>${RUN_STEPS}
e_low_N_high" >>${RUN_STEPS}
_merged_file" >>${RUN_STEPS}
tate_putative_mutation" >>${RUN_STEPS}
gene_annotation" >>${RUN_STEPS}
te_excel_sheet" >>${RUN_STEPS}
tate_low_tier_mutation" >>${RUN_STEPS}
fy_SJLLQ" >>${RUN_STEPS}


 created run_all_somatic_SNV_steps" >> $


LEVEL -gt 0 ]

 created run_all_somatic_SNV_steps" >> $
```

Another Example Script

Here's another example script that removes all occurrences of a specified word from a file:

```
root@DESKTOP-6872CI7:/home# cat > DeleteMatching.sh
echo enter file name
read file
echo enter word
read word
echo file before removing $word:
cat $file
grep -v -i $word $file > test
mv test $file
echo file after removing $word:
```

This script prompts the user to enter a filename and a word to remove. It then uses the grep command to search for lines in the file that do not contain the specified word and saves the result to a temporary file. Finally, it renames the temporary file to the original filename, effectively replacing the original file with the modified version.

Conclusion

Remember to always test your scripts thoroughly before using them in a production environment. With practice, you'll be able to create powerful and efficient shell scripts to automate your tasks and streamline your workflow. Happy scripting!