🚀 Bash Scripting for 42 Students

Your Complete Automation Toolkit

Stop typing the same commands over and over. Let's automate your 42 life!

🤔 Why Should You Care?

Right now, you probably do this every day:

```
gcc -Wall -Wextra -Werror *.c -o program
./program
git add .
git commit -m "fixed something"
git push
norminette *.c *.h
make fclean && make
```

🤔 Why Should You Care? (cont.)

What if I told you this could be just:

```
compile_and_test    # One command instead of 6!
```

That's the power of Bash scripting. And I've already built these scripts for you! 🎁

📦 What You're Getting

I've created a complete toolkit of scripts that solve real 42 problems. You'll learn by using them, then understanding how they work.

📁 Your new superpowers:

- `push` - Never type git commands again
- `cleaner` - Free up disk space instantly
- `check_forbidden` - Avoid -42 for forbidden functions

📦 What You're Getting (cont.)

More powerful tools:

- `push_swap_tester` - Test your push_swap like a pro
- `runner` - Valgrind made easy
- `git_helper` - Interactive git menu
- `todo_finder` - Find all TODO comments
- `norm_dir` - Auto-format your code

> 💡 **Promise**: By the end, you'll save 30+ minutes daily and never lose points for silly mistakes.

🏁 Level 1: Get Started (5 minutes)

Install Your Toolkit

```
# Get the scripts
git clone https://github.com/Moe-Salim91156/helper_scripts.git ~/helper_scripts

# Make them available everywhere
echo 'export PATH="$HOME/helper_scripts:$PATH"' >> ~/.zshrc
source ~/.zshrc

# Test it works
push --help
```

Your First Win: Never Type Git Commands Again

Instead of this pain:

```
git add .
git commit -m "working on libft"
git push
```

Just do this:

```
push
# It will ask for your commit message, then do everything!
```

How the Push Script Works

Let's look at the code:

```bash
#!/bin/bash
read -p "Commit message: " COMMIT_MSG
git add .
git commit -m "$COMMIT_MSG"
git push
```

✅ **Your first automation win!** You just saved 3 commands every single push.

🎯 Try It Now

1. Go to any git repository
2. Make a small change
3. Type `push`
4. Enter your commit message
5. Watch the magic! ✨

🧠 Level 2: Understanding Variables

You just used a script with variables! Let's understand what happened:

```
read -p "Commit message: " COMMIT_MSG  # Get user input
git commit -m "$COMMIT_MSG"            # Use that input
```

Variable Basics

```bash
#!/bin/bash
your_name="put your name here"
project="libft"
echo "Hi $your_name! Working on $project"
```

Command Line Arguments

```bash
echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
```

Your Next Power-Up: `cleaner`

Remember how your disk space is always full at 42? This script fixes that:

```
cleaner ~/42projects
```

What it does:

- Takes a directory path as argument (`$1`)
- Removes all cache folders inside
- Frees up gigabytes of space

How Cleaner Works

The magic behind it:

```
#!/bin/bash
TARGET_DIR="$1"  # Your directory argument

if [ -z "$TARGET_DIR" ]; then
    echo "Usage: cleaner /path/to/directory"
    exit 1
fi

echo "🧹 Cleaning cache folders in $TARGET_DIR..."
# ... cleaning magic happens here
```

💾 **Disk Space Saved**: Students report saving 2-5GB with one command!

🔀 Level 3: Making Decisions

Your `cleaner` script makes decisions - it checks if you provided a directory. Let's understand how:

```
if [ -z "$TARGET_DIR" ]; then    # If directory is empty
    echo "Error: Please provide a directory"
    exit 1
fi

if [ ! -d "$TARGET_DIR" ]; then  # If directory doesn't exist
    echo "Error: Directory not found"
    exit 1
fi
```

Your Next Superpower: `check_forbidden`

Never get -42 for forbidden functions again:

```
check_forbidden
```

What you'll see:

```
Checking for: printf
src/main.c:15:    printf("Hello World");  # ⚠️ Found forbidden function!

Checking for: execve
No matches found ✅
```

How check_forbidden Works

```bash
#!/bin/bash
forbidden_funcs=("printf" "execve" "system")

for func in "${forbidden_funcs[@]}"; do
    echo "Checking for: $func"
    grep -rnw . -e "$func"
done
```

🛡️ **Protection**: Run this before every push to avoid evaluation disasters!

🔄 Level 4: Loops and Arrays

Your `check_forbidden` script uses arrays and loops. Let's break it down:

```
# Array of forbidden functions
forbidden_funcs=("printf" "execve" "system")

# Loop through each function
for func in "${forbidden_funcs[@]}"; do
    echo "Checking for: $func"
    grep -rnw . -e "$func"  # Search in all files
done
```

Loop Types You'll Use

```
# Loop through files
for file in *.c; do
    echo "Compiling: $file"
    gcc -c "$file"
done

# Number ranges
for i in {1..5}; do
    echo "Test case $i"
done
```

Loop Types You'll Use (cont.)

```
# While loops
count=1
while [ $count -le 5 ]; do
    echo "Attempt $count"
    ((count++))
done
```

Your Code Detective: `todo_finder`

Find all your TODO comments instantly:

```
todo_finder
```

Output:

```
📄 src/libft.c:42:    // TODO: Optimize this function
📄 src/ft_split.c:15: // FIXME: Handle edge case
```

How todo_finder Works

The detective work:

```
find . -name "*.c" -o -name "*.h" | while read -r file; do
    grep -nE "TODO|FIXME" "$file" | while read -r line; do
        echo "📄 $file:$line"
    done
done
```

🕵️ **Never lose track**: Perfect for code reviews and tracking unfinished work!

🎮 Level 5: Interactive Menus

Time to level up with interactive scripts! Your `git_helper` gives you a menu:

```
git_helper
```

What you get:

```
Git Helper Menu:
1) Status
2) Add all
3) Commit
4) Push
5) Pull
6) Exit
Choose [1-5]:
```

The Menu Magic

```
while true; do
    echo "Git Helper Menu:"
    echo "1) Status  2) Add all  3) Commit  4) Push  5) Exit"
    read -p "Choose [1-5]: " choice

    case $choice in
        1) git status ;;
        2) git add . ;;
        3) read -p "Message: " msg; git commit -m "$msg" ;;
        4) push ;;  # Calls your push script!
        5) exit 0 ;;
        *) echo "Invalid option" ;;
    esac
done
```

🎯 **One interface**: All your git operations in one place, no commands to remember!

🧪 Level 6: Testing Like a Pro

Your biggest challenge at 42? Testing your projects properly. I've got you covered.

For push_swap: `push_swap_tester`

```
push_swap_tester
```

What push_swap_tester Does

What happens:

```
=== Testing Arrays of Size 3 ===
Test case: 3 2 1
Operations: 2
✅ Sorted correctly!

Test case: -1 5 4
Operations: 3
✅ Sorted correctly!

=== Testing Arrays of Size 100 ===
Test case: [random 100 numbers]
Operations: 547
✅ Sorted correctly!
```

The Testing Brain (Simplified)

```
test_case() {
    array=$1

    # Get your program's output
    INSTRUCTIONS=$(./push_swap "$array")

    # Check if it actually sorts
    RESULT=$(./checker_linux "$array" <<< "$INSTRUCTIONS")

    if [ "$RESULT" = "OK" ]; then
        echo "✅ Sorted correctly!"
    else
        echo "❌ Failed!"
    fi
}
```

🏆 **Professional testing**: Tests multiple sizes, counts operations, handles timeouts!

🛡️ Level 7: Debugging and Memory

Memory leaks driving you crazy? Meet your new best friend: `runner`

```
runner
```

Interactive Valgrind with runner

What you see:

```
42 Code Runner - Interactive Script
➤➤➤ Are you in the executable directory? (y/n)
>> y
➤➤➤ Provide the name of the executable:
>> ./push_swap
➤➤➤ Run full Valgrind check? (y/n)
>> y
➤➤➤ Arguments:
>> 4 67 3 87 23

Running Valgrind...
[Valgrind output shows your memory leaks]
```

Smart automation:

- Remembers your settings
- Handles different project structures
- Full or quick Valgrind modes
- No more typing long Valgrind commands!

🎨 Level 8: Code Formatting

Tired of norminette errors? `norm_dir` fixes everything:

```
norm_dir
```

What it does:

```
Put dir you want to norminette (press Enter for current directory):
Norminetting dir: /home/yourname/libft
[Formats all .c and .h files automatically]
```

The Formatting Magic

```
read -p "Directory (Enter for current): " INPUT_DIR
NORM_DIR=${INPUT_DIR:-$(pwd)}  # Use current if empty

find "$NORM_DIR" -name "*.c" -o -name "*.h" | \
    xargs python3 -m c_formatter_42
```

🎨 **Perfect formatting**: Never worry about spaces, tabs, or line length again!

🚀 Your Daily 42 Workflow

Here's how these scripts transform your day:

Morning Setup

```
cd ~/my_project
git_helper          # Quick git status check
todo_finder        # See what you need to do today
```

While Coding

```
check_forbidden     # Before you go too far
norm_dir           # Keep code clean as you go
```

🚀 Your Daily 42 Workflow (cont.)

Testing Phase

```
runner              # Test with Valgrind
push_swap_tester    # Specific project testing
```

End of Day

```
push                # Quick commit and push
cleaner ~/projects # Clean up disk space
```

Time saved daily: 30+ minutes
Frustration avoided: Priceless

🔧 Level 9: Customize Your Tools

These scripts are yours now! Make them better:

Easy Customizations

1. Update hardcoded paths:

```
# In cleaner script, change:
rm -rf /home/msalim/.cache
# To:
rm -rf /home/$USER/.cache
```

🔧 Level 9: Customize Your Tools (cont.)

2. Add more forbidden functions:

```
# In check_forbidden, add more:
forbidden_funcs=("printf" "execve" "system" "malloc" "free")
```

3. Improve the push script:

```
#!/bin/bash
# Check if in git repo
if [ ! -d ".git" ]; then
    echo "❌ Not a git repository!"
    exit 1
fi

read -p "Commit message: " COMMIT_MSG
git add .
git commit -m "$COMMIT_MSG"
git push
echo "✅ Successfully pushed!"
```

Create Your Own Scripts

Project compiler:

```
#!/bin/bash
echo "🔨 Compiling your project..."
make re
if [ $? -eq 0 ]; then
    echo "✅ Compilation successful!"
    ./your_program
else
    echo "❌ Compilation failed!"
fi
```

🎓 Level Up: Advanced Tips

Script Chaining

Make scripts call each other:

```
#!/bin/bash
check_forbidden  # Check for forbidden functions
norm_dir        # Format code
push            # Commit and push
```

Error Handling

Make your scripts robust:

```
#!/bin/bash
set -e  # Exit on any error

if [ ! -f "Makefile" ]; then
    echo "❌ No Makefile found!"
    exit 1
fi

make
echo "✅ Build successful!"
```

🎓 Level Up: Advanced Tips (cont.)

Debugging Scripts

When something goes wrong:

```
bash -x your_script.sh  # Shows every command
```

🆘 Troubleshooting

Common Issues

"Permission denied"

```
chmod +x script_name
```

"Command not found"

```
# Make sure PATH is set:
echo $PATH
# Should include /home/yourname/helper_scripts
```

Script doesn't work as expected

```
# Debug mode:
bash -x script_name
```

🆘 Troubleshooting (cont.)

Getting Help

- Read the script code - it's educational!
- Test on dummy projects first
- Ask fellow students who use the scripts
- DM me on Slack: `msalim`

🏆 Your Bash Scripting Journey

Week 1: Get Comfortable

- [ ] Install the scripts
- [ ] Use `push` daily
- [ ] Try `cleaner` when disk is full
- [ ] Run `check_forbidden` before pushes

Week 2: Explore More

- [ ] Use `git_helper` for all git operations
- [ ] Try `runner` for memory testing
- [ ] Use `todo_finder` for code reviews
- [ ] Format code with `norm_dir`

🏆 Your Bash Scripting Journey (cont.)

Week 3: Understand & Customize

- [ ] Read through script code
- [ ] Make small modifications
- [ ] Update hardcoded paths
- [ ] Add your own test cases

Week 4: Create Your Own

- [ ] Write a project-specific script
- [ ] Combine multiple scripts
- [ ] Share improvements with classmates
- [ ] Contribute back to the repo!

🎯 Mission Complete!

You started here: Typing the same commands repeatedly, getting frustrated with norminette, forgetting to check forbidden functions.

You are now: A bash scripting wizard with a complete automation toolkit, saving hours daily and never making silly mistakes again.

What You've Gained

⚡ **Speed**: Automated all repetitive tasks

🛡️ **Safety**: Never forget checks again

🧠 **Understanding**: Know how scripts work

🔧 **Customization**: Can modify and create scripts

💪 **Confidence**: Ready for any 42 project

Next Adventures

- Automate project-specific tasks
- Create test suites for your projects
- Share scripts with your peers (ME for example !)
- Build more complex automations

🤝 Join the Automation Revolution

Spread the word! Help your fellow 42 students and **RAAS MEMBERS**:

- Share these scripts in your clusters
- Teach others how to use them
- Contribute improvements to the repo
- Create project-specific versions

Remember: Every expert was once a beginner. You now have the tools and knowledge to automate your 42 journey!

📞 Stay Connected

- **GitHub**: My Github Account
- **Slack**: DM `msalim` for questions, suggestions, or just to say thanks!

🚀 *From one 42 student to another: Stop working harder, start working smarter!*

Happy scripting! 🎯✨

Done By Mohammad Salim
intra : msalim