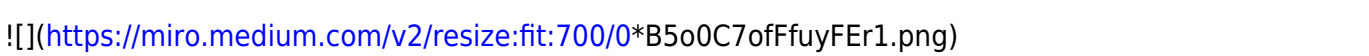
<!DOCTYPE markdown># Advanced WSL Configuration. Setting up WSL (Windows Subsystem for... | by Smit Gabani | Medium

Advanced WSL Configuration

(https://miro.medium.com/v2/resize:fill:32:32/1*Pj5qvbgXRYnqAz6okTkxng.jpeg)

6 min read

Jul 19, 2024


(https://miro.medium.com/v2/resize:fit:700/0*B5o0C7ofFfuyFEr1.png)

Setting up WSL (Windows Subsystem for Linux) is just the beginning. To enhance your development experience, it’s essential to configure WSL optimally. Advanced configuration can improve performance, customize your development environment, and streamline your workflow. Here’s a detailed guide on advanced WSL configuration.

Allocating Resources

WSL2 uses a lightweight virtual machine to run Linux. By default, it’s configured to use a dynamic amount of memory and CPU resources. You can specify resource limits in the WSL configuration file.

Create or Edit the `.wslconfig` File:

- Open Notepad or your preferred text editor. - Create a file named `.wslconfig` in your user profile directory (`C:\Users\<YourUsername>`).

Add Resource Limits:

\[wsl2\] memory\=4GB processors\=2

- **memory**: Sets the maximum amount of RAM for WSL2. - **processors**: Defines the number of CPU cores WSL2 can use.

Apply Changes:

- Restart WSL2 by running `wsl –shutdown` in PowerShell or CMD.

Using a Faster Filesystem

WSL2 uses a virtual hard disk (VHD) for its filesystem. Accessing files in the Linux filesystem can be slower than in the Windows filesystem. To improve speed:

Store Files on Windows Filesystem:

Access the Windows filesystem from WSL at `/mnt/c/` and store frequently used files here for better performance.

Use `wsl.conf` for Automount Settings:

- Create or edit `/etc/wsl.conf` in your WSL2 instance. - Add settings to control how the Windows filesystem is mounted:

\[automount\] options = “metadata,umask=22,fmask=11”

- This configuration improves compatibility and access permissions.

Customizing Your WSL Environment

Configuring Shell Preferences

Changing Default Shell:

- If you prefer a different shell like Zsh over Bash, you can install it and set it as the default. - Install Zsh:

sudo apt-get install zsh

- Set Zsh as default:

chsh -s $(which zsh)

Customizing Shell Prompt:

- Modify the prompt in your shell configuration file (`~/.bashrc`, `~/.zshrc`). - For example, to change the Bash prompt:

PS1\='\[\\u@\\h \\W\]\\$ '

Setting Up Aliases and Functions

Creating Aliases:

- Add custom aliases to your shell configuration file. - For example, add the following to `~/.bashrc`

alias ll='ls -la' alias gs='git status'

Here are some custom alias I use:

\# ~/.bash_aliases or ~/.zsh_aliases

\# List all files in long format, including hidden files alias ll='ls -la'

\# Display the status of the Git repository alias gs='git status'

\# List files with human-readable sizes alias lh='ls -lh'

\# Show hidden files in long format with human-readable sizes alias lsa='ls -la'

\# Find files by name alias f='find . -name'

\# Create a directory and change into it alias mkcd='mkdir -p $1 && cd $1'

\# Show the last commit log in Git alias gl='git log -1'

\# Show changes made in the last commit alias gcl='git show –stat'

\# Fetch changes from the origin and rebase alias gfr='git fetch origin && git rebase'

\# Reset local changes in Git alias grr='git reset –hard'

\# List Git branches with remote tracking alias gb='git branch -vv'

\# Show disk usage in human-readable format alias du\='du -h'

\# Show top processes alias top10='top -b -n 1 | head -n 20'

\# Check system uptime in a pretty format alias uptime\='uptime -p'

\# Display the last 10 system logs alias dmesg10='dmesg | tail -n 10'

\# Check external IP address alias myip='curl ifconfig.me'

\# Show network interfaces and their status alias netinfo='ip addr'

\# Ping a host with 4 packets alias pinghost='ping -c 4'

\# Open a file in the default editor (using Vim) alias edit='vim'

\# Search for a keyword in files with color highlighting alias grep='grep –color=auto'

\# Remove files with confirmation prompt alias rm\='rm -i'

\# Count lines, words, and characters in a file alias wc\='wc -l -w -c'

\# Start a local HTTP server using Python 3 alias serve='python3 -m http.server'

\# Clear the terminal screen alias cls='clear'

\# Show the most recent Git commits with format alias glf='git log –oneline –graph –decorate'

Load the Aliases:

- For `bash`, include the following line in your `~/.bashrc` file:

if \[-f ~/.bash_aliases \]; then

```
  . ~/.bash\_aliases
```

fi

- For `zsh`, include the following line in your `~/.zshrc` file:

if \[-f ~/.zsh_aliases \]; then

```
  . ~/.zsh\_aliases
```

fi

Apply Changes:

- Apply the changes by running:

source ~/.bashrc \# For bash source ~/.zshrc \# For zsh

Defining Functions:

- Add custom functions to automate repetitive tasks. - Example function to quickly navigate to a project directory:

function proj() {

```
  cd ~/projects/$1
```

}

Some functions I use

\# Switch to a specified Git branch with confirmation function gsw() {

```
  local branch=$1
  if \[ -z "$branch" \]; then
      echo "Usage: gsw <branch\_name>"
      return 1
  fi
  git checkout \\$branch
  if \[ $? -eq 0 \]; then
      echo "Switched to branch $branch"
  else
      echo "Failed to switch branch"
  fi
```

}

- **Usage**: `gsw feature_branch` switches to `feature_branch` and provides feedback. - **Benefit**: Adds error handling and user feedback.

\# Update and upgrade system packages in one command function update_system() {

```
  echo "Updating package lists..."
  sudo apt-get update
  echo "Upgrading packages..."
  sudo apt-get upgrade -y
  echo "System updated and upgraded."
```

}

- **Usage**: `update_system` performs both update and upgrade operations. - **Benefit**: Combines two common tasks into one command for convenience.

\# Backup a file or directory with a timestamp function backup() {

```
  local source\=$1
  local destination="\\$source\-$(date +%F-%T).bak"
  if \[ -z "$source" \]; then
```

```
        echo "Usage: backup <source>"
        return 1
    fi
    cp -r $source $destination
    echo "Backup of $source created at $destination"
```

}

- **Usage**: `backup important_file` creates a backup with a timestamp. - **Benefit**: Automatically timestamps backups, which helps in version control and recovery.

Installing Essential Tools

Package Managers:

- Install `apt` tools to manage packages. - For example, install `vim`, `htop`, and `curl`:

sudo apt-get install vim htop curl

Development Tools:

- Install additional tools as needed (e.g., `docker`, `node`, `python`).

Managing WSL Integration with Windows

Accessing Linux Files from Windows

Using File Explorer:

- Access Linux files from Windows via `\\wsl$`. - Open File Explorer and type `\\wsl$\` in the address bar.

Creating Shortcuts:

- Create Windows shortcuts to frequently accessed Linux directories.

Running Windows Applications from WSL

Accessing Windows Apps:

- You can run Windows executables directly from WSL. - Example:

/mnt/c/Windows/System32/notepad.exe

Setting Up Environment Variables:

- Add Windows environment variables to your WSL environment for seamless integration.

Backing Up and Restoring WSL

Exporting and Importing WSL Distributions

Export a WSL Distribution:

- Use `wsl –export` to create a backup of your distribution:

wsl –export Ubuntu ubuntu_backup.tar

Import a WSL Distribution:

- Restore from a backup using `wsl –import`:

wsl –import UbuntuNew C:\\path\\to\\install\\folder ubuntu_backup.tar

Regular Backups

Automate Backups:

Set up scripts to periodically backup your WSL environment.

Create a Backup Script

Create a Backup Script in WSL

Get Smit Gabani’s stories in your inbox

Join Medium for free to get updates from this writer.

Subscribe

Subscribe

Open your WSL terminal.

Create a script file for the backup, e.g., `backup_wsl.sh`.

nano ~/backup_wsl.sh

Add the following content to `backup_wsl.sh`. This script will create a compressed tarball of your home directory and save it to a backup location:

#!/bin/bash

\# Define backup location and filename BACKUP_DIR=“/mnt/c/Users/YourWindowsUsername/WSL_Backups” TIMESTAMP=$(date +%F-%H-%M-%S) BACKUP_FILE=“wsl_backup_$TIMESTAMP.tar.gz”

\# Ensure backup directory exists mkdir -p “$BACKUP_DIR”

\# Create a tarball of the home directory tar -cvzf “$BACKUP_DIR/$BACKUP_FILE” /home/yourusername

\# Print a message indicating backup completion echo “Backup completed: $BACKUP_DIR/$BACKUP_FILE”

Replace `YourWindowsUsername` with your actual Windows username and `yourusername` with your WSL username.

Make the script executable:

chmod +x ~/backup_wsl.sh

Schedule the Backup with Windows Task Scheduler

Create a Batch Script to Run the WSL Backup Script

Open Notepad or any text editor on Windows.

Create a batch file, e.g., `run_wsl_backup.bat`.

@echo off wsl bash -c "~/backup_wsl.sh"

Save the file as `run_wsl_backup.bat`.

Schedule the Task in Windows Task Scheduler

1. Open **Task Scheduler** from the Start menu. 2. Click on **Create Basic Task** in the Actions pane. 3. Name your task, e.g., "WSL Backup" and click **Next**. 4. Choose a trigger, e.g., **Daily** or **Weekly**, depending on how frequently you want to run the backup. Click **Next**. 5. Set the start time and recurrence settings as desired. Click **Next**. 6. Choose **Start a Program** as the action. Click **Next**. 7. Click **Browse** and select the `run_wsl_backup.bat` batch file you created. Click **Next**. 8. Review the settings and click **Finish**.

Verify and Test Your Setup

- **Manually Run the Task**: You can test the setup by right-clicking on the scheduled task in Task Scheduler and selecting **Run**. - **Check Backup Files**: Verify that the backup files are created in the specified backup directory (`C:\Users\YourWindowsUsername\WSL_Backups`).

Store Backups Securely:

- Use cloud storage or external drives for backup storage.

That's everything for now. Please share your feedback on this article by commenting on your preferences, any areas of concern, and any questions you may have.