tmux Mastery Guide

A Comprehensive Practical Guide for Ubuntu 24.04.01

Solving SSH Session Persistence for Long-Running Tasks

Prepared by Z.ai

Table of Contents

1. Introduction to tmux - Understanding terminal multiplexing

2. Installation on Ubuntu 24.04.01 - Getting started with the latest version

3. Core Concepts - Sessions, windows, and panes explained

4. The SSH Persistence Problem - Why your tasks die and how tmux solves it

5. Essential Commands Reference - Complete command cheat sheet

6. Key Bindings Quick Reference - Keyboard shortcuts for efficiency

7. Practical SSH Workflows - Real-world session management examples

8. Configuration and Customization - Optimizing tmux for your workflow

9. Plugin Ecosystem - TPM, tmux-resurrect, and tmux-continuum

10. tmux vs GNU Screen - Comparison and migration guide

11. Advanced Automation - Scripting and automation techniques

12. Troubleshooting Guide - Common issues and solutions

1. Introduction to tmux

tmux (terminal multiplexer) is a powerful command-line tool that revolutionizes how developers and system administrators work with terminal sessions. At its core, tmux enables you to create, manage, and switch between multiple terminal sessions within a single terminal window. More importantly, tmux sessions persist independently of your terminal connection, making it an indispensable tool for remote work and long-running tasks on cloud servers.

1.1 What is a Terminal Multiplexer?

A terminal multiplexer is a software application that allows you to split a single physical terminal into multiple virtual terminals. Before terminal multiplexers existed, developers were limited to one terminal session per physical terminal window. This meant if you wanted to run multiple commands simultaneously or keep a process running while working on something else, you needed multiple terminal windows or physical machines.

tmux solves this elegantly by creating a layer between your terminal emulator and the shell processes. When you run tmux, it creates a server process that manages all your sessions, windows, and panes. Your terminal becomes a client that connects to this server. This client-server architecture is what enables session persistence - even if your SSH connection drops or you close your terminal, the tmux server continues running with all your processes intact.

1.2 Key Features and Benefits

tmux offers several compelling features that make it essential for modern development workflows:

Feature	Description	Benefit
Session Persistence	Sessions survive disconnections	Long-running tasks continue uninterrupted
Window Management	Multiple windows per session	Organize related tasks together
Pane Splitting	Split windows horizontally/vertically	View multiple terminals simultaneously
Detachable Sessions	Detach and reattach at will	Resume work from any location
Scriptability	Full command-line control	Automate complex workflows
Cross-Platform	Works on Linux, macOS, BSD	Consistent experience everywhere

Table 1: Key tmux features and their practical benefits

2. Installation on Ubuntu 24.04.01

Ubuntu 24.04.01 LTS includes tmux in its default repositories, making installation straightforward. The version included has been thoroughly tested and is fully compatible with all features discussed in this guide.

2.1 Standard Installation

Install tmux using the apt package manager:

```
sudo apt update && sudo apt install tmux -y
```

This command updates your package index and installs tmux along with its dependencies. The installation typically completes within seconds on modern systems.

2.2 Verifying Installation

After installation, verify that tmux is working correctly:

```
tmux -V
```

This should display the installed version number. Ubuntu 24.04.01 typically ships with tmux 3.4 or later, which includes all modern features including true color support and improved copy mode.

2.3 Building from Source (Optional)

If you need the absolute latest version or specific features not included in the repository version, you can build tmux from source. This requires installing build dependencies and cloning the repository:

```
sudo apt install build-essential automake libevent-dev ncurses-dev bison pkg-config
```

```
git clone https://github.com/tmux/tmux.git && cd tmux && sh autogen.sh && ./configure
&& make && sudo make install
```

3. Core Concepts

Understanding tmux requires grasping its hierarchical structure. tmux organizes terminal sessions into three levels: sessions contain windows, and windows contain panes. This section explains each level in detail.

3.1 Sessions

A session is the top-level container in tmux. Think of a session as a complete workspace dedicated to a specific project or task. Each session maintains its own set of windows and remembers which window was active when you last detached. Sessions are independent entities - you can have one session for development, another for system monitoring, and a third for database work. Each session runs its own server process, ensuring that work in one session is completely isolated from others.

When you SSH into a remote server and start a tmux session, all processes within that session continue running even after you disconnect. This is the fundamental feature that solves the SSH persistence problem. Sessions can be named for easy identification, making it simple to manage multiple concurrent workspaces.

3.2 Windows

Windows are the second level in the hierarchy. Each session can contain multiple windows, similar to tabs in a web browser. Windows within a session can be navigated using keyboard shortcuts, allowing you to quickly switch

between different contexts without leaving your session. Each window maintains its own command history and can contain multiple panes. You can rename windows to reflect their purpose, such as "editor", "logs", or "database", helping you stay organized across complex projects.

3.3 Panes

Panes are the most granular level in tmux. A pane represents an individual terminal within a window. You can split windows both horizontally and vertically to create multiple panes, allowing you to view and interact with several terminals simultaneously. This is particularly useful for monitoring logs while editing code, or running tests while viewing their output. Panes can be resized, moved between windows, and even broken out into their own windows.

Level	Container	Contains	Analogy
1 (Highest)	Session	Multiple Windows	Browser window
2	Window	Multiple Panes	Browser tab
3 (Lowest)	Pane	Single terminal/shell	Split view panel

Table 2: tmux hierarchical structure

4. The SSH Persistence Problem

Your primary concern with SSH sessions being fragile is shared by virtually every developer and system administrator who works with remote servers. This section explains why SSH sessions disconnect and how tmux provides a robust solution.

4.1 Why SSH Sessions Terminate

SSH connections can be interrupted for numerous reasons, and understanding these causes helps you implement appropriate safeguards:

Network Instability: Temporary network disruptions, ISP routing changes, or WiFi signal fluctuations can cause SSH connections to drop unexpectedly. Even brief interruptions of a few seconds can terminate your session and all running processes.

Timeout Configuration: Many network administrators configure firewalls and routers to terminate idle connections. If your SSH session remains idle for too long (often 15-30 minutes), the connection may be automatically closed by intermediate network devices.

Client-Side Issues: Closing your laptop lid, putting your computer to sleep, or accidentally closing the terminal window will immediately terminate all SSH processes running in that session.

Server-Side Reboots: System updates, maintenance windows, or crashes on the remote server will end all SSH sessions. While tmux cannot protect against server reboots, plugins like tmux-resurrect can help recover your workspace after the server comes back online.

4.2 How tmux Solves This Problem

tmux addresses SSH session fragility through its client-server architecture. When you start tmux, it creates a server process that runs independently of your SSH connection. Your terminal becomes a client that attaches to this server. When you detach from a session or when your SSH connection drops, the tmux server continues running with all your processes intact. You can later reconnect to the same session from any location and resume exactly where you left off.

4.3 The SSH + tmux Workflow

Here is the recommended workflow for working with remote servers using SSH and tmux:

Step 1: SSH into your remote server: ssh user@remote-host

Step 2: Start or attach to a tmux session: tmux new -s myproject OR tmux attach -t myproject

Step 3: Run your long-running task: ./build.sh, python train.py, npm run build, etc.

Step 4: Detach from tmux (NOT the SSH session): Press Ctrl+b, then d

Step 5: Disconnect SSH or close your laptop: Your task continues running

Step 6: Reconnect later: SSH back to the server

Step 7: Reattach to your session: tmux attach -t myproject

This workflow ensures your tasks run uninterrupted regardless of network conditions or client-side issues. The only way your processes will stop is if the remote server itself is rebooted or experiences a crash.

5. Essential Commands Reference

This section provides a comprehensive reference of tmux commands organized by category. These commands can be run from the shell when tmux is not in a command mode, or prefixed with Ctrl+b : to enter command mode.

5.1 Session Management Commands

Command	Description	Example
tmux new -s name	Create a new named session	tmux new -s webapp
tmux new -s name -d	Create session detached (background)	tmux new -s build -d
tmux ls	List all active sessions	tmux ls
tmux attach -t name	Attach to specific session	tmux attach -t webapp
tmux attach	Attach to last used session	tmux attach
tmux detach	Detach from current session	tmux detach
tmux kill-session -t name	Terminate a specific session	tmux kill-session -t old
tmux kill-server	Kill all sessions and stop server	tmux kill-server
tmux rename -t old new	Rename a session	tmux rename -t 0 main

Table 3: Session management commands

5.2 Window Management Commands

Command	Description
tmux new-window -n name	Create new window with name
tmux select-window -t name	Switch to named window
tmux next-window	Move to next window
tmux previous-window	Move to previous window
tmux kill-window -t name	Close specified window
tmux rename-window name	Rename current window

Table 4: Window management commands

5.3 Pane Management Commands

Command	Description
tmux split-window -h	Split pane vertically (left/right)
tmux split-window -v	Split pane horizontally (top/bottom)

tmux select-pane -U/D/L/R	Move to pane Up/Down/Left/Right
tmux swap-pane -U/D	Swap pane with neighbor
tmux kill-pane	Close current pane
tmux break-pane	Move pane to its own window
tmux join-pane -t window	Move pane to another window
tmux resize-pane -U 10	Resize pane by 10 cells up

Table 5: Pane management commands

6. Key Bindings Quick Reference

tmux uses a prefix key system. By default, the prefix is Ctrl+b (often written as C-b). You press the prefix first, release it, and then press the command key. This section provides a comprehensive reference of all default key bindings, which will become second nature with practice.

6.1 Essential Session Bindings

Key Sequence	Action	Description
Ctrl+b d	Detach	Detach from session (keeps it running)
Ctrl+b s	List sessions	Show all sessions with interactive list
Ctrl+b $	Rename session	Prompt for new session name
Ctrl+b (	Previous session	Switch to previous session
Ctrl+b)	Next session	Switch to next session

Table 6: Session key bindings

6.2 Window Bindings

Key Sequence	Action	Description
Ctrl+b c	Create window	Create a new window
Ctrl+b n	Next window	Move to next window
Ctrl+b p	Previous window	Move to previous window
Ctrl+b 0-9	Select window	Switch to window by number
Ctrl+b &	Kill window	Close current window after confirmation
Ctrl+b ,	Rename window	Prompt for new window name
Ctrl+b w	Window list	Interactive window selection

Table 7: Window key bindings

6.3 Pane Bindings

Key Sequence	Action	Description
Ctrl+b %	Split vertical	Split pane into left/right
Ctrl+b "	Split horizontal	Split pane into top/bottom
Ctrl+b Arrow	Navigate panes	Move to pane in arrow direction
Ctrl+b o	Next pane	Cycle through panes

Ctrl+b x	Kill pane	Close current pane
Ctrl+b z	Zoom pane	Toggle pane full-screen
Ctrl+b q	Show pane numbers	Display pane numbers for selection
Ctrl+b {	Swap pane up	Swap with previous pane
Ctrl+b }	Swap pane down	Swap with next pane
Ctrl+b !	Break pane	Move pane to new window

Table 8: Pane key bindings

6.4 Copy Mode Bindings

Copy mode allows you to navigate, search, and copy text from the terminal scrollback buffer. This is essential for extracting output from long-running commands or reviewing logs.

Key Sequence	Action	Description
Ctrl+b [	Enter copy mode	Navigate scrollback buffer
Ctrl+b]	Paste buffer	Paste copied text
Ctrl+b #	List buffers	Show all paste buffers
= (in copy mode)	List buffers	Interactive buffer selection

Table 9: Copy mode key bindings

Navigation in Copy Mode (vi-style):

```
k/j - Move up/down one line
h/l - Move left/right one character
Ctrl+b / Ctrl+f - Page up / Page down
g / G - Go to top / bottom of buffer
/ - Start forward search
? - Start backward search
Space - Start selection
Enter - Copy selection and exit
q - Exit copy mode
```

7. Practical SSH Workflows

This section demonstrates real-world workflows for using tmux with SSH, specifically addressing your concern about long-running tasks being terminated due to connection issues.

7.1 Basic Persistent Development Session

This workflow establishes a simple but effective development environment that persists across SSH sessions:

```
# Connect to remote server
ssh developer@myserver.example.com

# Check for existing sessions
tmux ls

# If no session exists, create one
tmux new -s dev

# Or attach to existing session
tmux attach -t dev
```

Once inside your session, you can split panes for different tasks. A common setup includes one pane for editing code, another for running tests or builds, and a third for monitoring logs. When you need to leave, simply press Ctrl+b d to detach. Your work continues uninterrupted.

7.2 Long-Running Build or Deployment

For tasks like compiling large projects, training machine learning models, or deploying applications, follow this workflow:

```
# Create a dedicated session for the build
tmux new -s build

# Start your long-running task
./build.sh 2>&1 | tee build.log

# Detach and let it run: Ctrl+b d

# Later, reattach to check progress
tmux attach -t build
```

Using tee ensures you have a log file even if something unexpected happens. You can detach immediately after starting the command and check back hours or days later.

7.3 Multi-Server Monitoring Dashboard

tmux excels at creating monitoring dashboards. This example shows how to monitor multiple servers:

```
# Create monitoring session
tmux new -s monitor

# Split into four panes (Ctrl+b % then Ctrl+b " for each)
# Pane 1: System resources
htop

# Pane 2: Application logs
tail -f /var/log/app/app.log

# Pane 3: Nginx access logs
tail -f /var/log/nginx/access.log

# Pane 4: Database monitoring
watch -n 5 "mysql -e 'SHOW PROCESSLIST;'"
```

This dashboard persists even if you close your laptop. Simply reattach later to continue monitoring. For production environments, consider using tmux-continuum to automatically save and restore this layout.

7.4 SSH Config Integration

You can configure SSH to automatically start or attach to tmux sessions when connecting to specific hosts. Add this to your local ~/.ssh/config file:

```
Host myserver
HostName server.example.com
User developer
RequestTTY yes
RemoteCommand tmux attach -t main 2>/dev/null || tmux new -s main
```

With this configuration, running "ssh myserver" automatically attaches to your main session or creates one if it does not exist. This creates a seamless workflow where you never have to think about session management.

7.5 Quick Reference: One-Line Commands

Here are useful one-liners for common SSH + tmux scenarios:

Scenario	Command
SSH + attach or create	ssh host -t "tmux attach || tmux new"
Create detached session with command	tmux new -s name -d "long-command"
Send command to running session	tmux send-keys -t session "cmd" Enter
Kill all sessions except current	tmux kill-session -a
Save session layout to file	tmux list-windows -a > layout.txt

Table 10: Quick reference commands for SSH + tmux workflows

8. Configuration and Customization

tmux configuration is stored in ~/.tmux.conf. This file allows you to customize key bindings, appearance, behavior, and much more. This section covers essential configurations for improving your tmux experience.

8.1 Essential Configuration Options

Create or edit ~/.tmux.conf with these recommended settings:

```
# Change prefix from Ctrl+b to Ctrl+a (easier to reach)
set -g prefix C-a
unbind C-b
bind C-a send-prefix

# Enable mouse support (click to select pane, scroll)
set -g mouse on

# Start window numbering at 1 (easier to reach)
set -g base-index 1
setw -g pane-base-index 1

# Automatically renumber windows when one is closed
set -g renumber-windows on

# Increase scrollback buffer size (default is 2000)
set -g history-limit 50000

# Enable 256 colors and true color support
set -g default-terminal "screen-256color"
set -ga terminal-overrides ",*256col*:Tc"

# Reduce delay for escape key (better for vim)
set -sg escape-time 0

# Split panes using | and - (more intuitive)
bind | split-window -h -c "#{pane_current_path}"
bind - split-window -v -c "#{pane_current_path}"
unbind \"
unbind %
```

8.2 Status Bar Customization

The status bar provides valuable information at a glance. Here is a customized status bar configuration:

```
# Status bar position and refresh rate
set -g status-position bottom
set -g status-interval 5

# Status bar colors
set -g status-style bg=black,fg=white

# Left side: session name
set -g status-left " #S #[default]"
set -g status-left-length 20

# Right side: datetime
set -g status-right " %H:%M %Y-%m-%d "

# Window list format
set -g window-status-format " #I:#W "
set -g window-status-current-format " #I:#W "
set -g window-status-current-style bg=blue,fg=white,bold
```

8.3 More Intuitive Pane Navigation

Use Alt+Arrow keys to navigate between panes without the prefix key:

```
# Navigate panes with Alt+Arrow
bind -n M-Left select-pane -L
bind -n M-Right select-pane -R
bind -n M-Up select-pane -U
bind -n M-Down select-pane -D
```

8.4 Reloading Configuration

After editing your configuration, reload it without restarting tmux:

```
bind r source-file ~/.tmux.conf \; display "Config reloaded!"
```

With this binding, press Ctrl+b r (or Ctrl+a r if you changed the prefix) to reload your configuration.

9. Plugin Ecosystem

tmux has a rich plugin ecosystem that extends its functionality. The most important plugins for SSH session persistence are tmux-resurrect and tmux-continuum, which together provide automatic session saving and restoration across server reboots.

9.1 TPM (Tmux Plugin Manager)

TPM is the standard plugin manager for tmux. Install it first:

```
# Clone TPM to your home directory
git clone https://github.com/tmux-plugins/tpm ~/.tmux/plugins/tpm

# Add this to the BOTTOM of ~/.tmux.conf:
# List of plugins
set -g @plugin "tmux-plugins/tpm"
set -g @plugin "tmux-plugins/tmux-sensible"

# Initialize TPM (keep this at the bottom)
run "~/.tmux/plugins/tpm/tpm"

# Install plugins: Press prefix + I (capital i)
```

9.2 tmux-resurrect: Manual Session Saving

tmux-resurrect saves your tmux environment (sessions, windows, panes, and even running programs) to disk. This means you can restore your complete workspace even after a server reboot.

```
# Add to your plugin list in ~/.tmux.conf:
set -g @plugin "tmux-plugins/tmux-resurrect"

# Key bindings after installation:
# prefix + Ctrl-s - Save session
# prefix + Ctrl-r - Restore session
```

The saved data includes the layout of all panes, their current directories, and even running programs (with some limitations). This is invaluable for complex development setups that take time to configure.

9.3 tmux-continuum: Automatic Session Saving

tmux-continuum builds on tmux-resurrect by adding automatic saving. It saves your sessions every 15 minutes (configurable) and can automatically restore them when tmux starts.

```
# Add to your plugin list:
set -g @plugin "tmux-plugins/tmux-continuum"

# Optional: Change save interval (default 15 minutes)
set -g @continuum-save-interval "10"

# Optional: Auto-restore on tmux start
set -g @continuum-restore "on"
```

With both plugins installed, your sessions are automatically backed up and can be restored after any interruption, including server reboots. This essentially solves the SSH persistence problem completely, as your workspace state is

continuously preserved.

9.4 Other Useful Plugins

Plugin	Purpose	Key Feature
tmux-sensible	Sensible defaults	Better default settings
tmux-pain-control	Pane management	Easy pane resize/move
tmux-yank	Copy to clipboard	System clipboard integration
tmux-open	Open URLs	Open selection in browser
tmux-copycat	Enhanced search	Regex search in copy mode
tmux-logging	Logging	Save pane output to file

Table 11: Popular tmux plugins

10. tmux vs GNU Screen

GNU Screen is the predecessor to tmux and still has users who prefer its simplicity. This section compares both tools to help you understand why tmux is generally recommended for new users while acknowledging situations where Screen might still be appropriate.

10.1 Feature Comparison

Feature	tmux	GNU Screen
Architecture	Client-server model	Client-server model
Pane Splitting	Native horizontal/vertical	Requires patches for vertical
Configuration	Clean, documented syntax	Cryptic syntax
Scriptability	Excellent (full control)	Limited
Plugin System	Yes (TPM)	No
Active Development	Very active	Minimal (maintenance)
Default Prefix	Ctrl+b	Ctrl+a
Session Persistence	Excellent (with plugins)	Basic
Documentation	Comprehensive man page	Limited

Table 12: Feature comparison between tmux and GNU Screen

10.2 When to Choose Each

Choose tmux when:

- You need advanced pane management and window layouts

- You want automatic session saving and restoration

- You prefer extensive customization and plugins

- You script terminal workflows

- You want an actively developed tool with modern features

Choose GNU Screen when:

- You only need basic session persistence

- Screen is pre-installed and you cannot install tmux

- You are already familiar with Screen and prefer its key bindings

- You work on very old systems where tmux is unavailable

10.3 Migration Guide (Screen to tmux)

If you are migrating from GNU Screen, this mapping of equivalent commands will help:

Screen Command	tmux Equivalent	Action
screen -S name	tmux new -s name	Create named session
screen -ls	tmux ls	List sessions
screen -r name	tmux attach -t name	Attach to session
Ctrl+a d	Ctrl+b d	Detach session
Ctrl+a c	Ctrl+b c	Create window
Ctrl+a n	Ctrl+b n	Next window
Ctrl+a S	Ctrl+b "	Split horizontal
Ctrl+a |	Ctrl+b %	Split vertical (screen needs patch)

Table 13: Screen to tmux command migration guide

11. Advanced Automation

tmux can be scripted to automate complex workspace setups. This is invaluable for developers who want consistent environments across projects or for system administrators managing multiple monitoring dashboards.

11.1 Scripting Session Creation

Create a script to set up a complete development environment:

```
#!/bin/bash
# dev-session.sh - Create a development session

SESSION="dev"

# Kill existing session if present
tmux kill-session -t $SESSION 2>/dev/null

# Create new session with first window named "editor"
tmux new-session -d -s $SESSION -n editor

# Send commands to the editor window
tmux send-keys -t $SESSION:editor "cd ~/project && vim" Enter

# Create window for running tests
tmux new-window -t $SESSION -n tests
```

```
tmux send-keys -t $SESSION:tests "cd ~/project && npm test" Enter

# Create window for git operations
tmux new-window -t $SESSION -n git
tmux send-keys -t $SESSION:git "cd ~/project && git status" Enter

# Create window for logs with split panes
tmux new-window -t $SESSION -n logs
tmux split-window -h -t $SESSION:logs
tmux send-keys -t $SESSION:logs.0 "tail -f logs/app.log" Enter
tmux send-keys -t $SESSION:logs.1 "tail -f logs/error.log" Enter

# Select the editor window
tmux select-window -t $SESSION:editor

# Attach to the session
tmux attach -t $SESSION
```

11.2 Project-Specific Setup Script

Save this script in your project directory and run it to create a consistent workspace:

```
#!/bin/bash
# setup-tmux.sh

PROJECT_NAME=${PWD##*/}
SESSION="proj-$PROJECT_NAME"

tmux has-session -t $SESSION 2>/dev/null

if [ $? != 0 ]; then
tmux new-session -d -s $SESSION -n main
tmux split-window -v -t $SESSION
tmux split-window -h -t $SESSION:0.0
tmux send-keys -t $SESSION:0.0 "vim" Enter
tmux send-keys -t $SESSION:0.1 "npm run dev" Enter
tmux send-keys -t $SESSION:0.2 "git status" Enter
fi

tmux attach -t $SESSION
```

11.3 Using tmuxinator

tmuxinator is a Ruby gem that provides a declarative YAML syntax for defining tmux sessions. It offers a more maintainable alternative to shell scripts:

```
# Install tmuxinator
gem install tmuxinator

# Create a project configuration
mux new myproject

# Example ~/.tmuxinator/myproject.yml:
name: myproject
root: ~/projects/myproject
windows:
- editor: vim
- server: npm run dev
- logs: tail -f logs/development.log
- database: psql -d myproject_development
```

Start the project with "mux start myproject" and tmuxinator handles all the tmux commands internally.

12. Troubleshooting Guide

This section addresses common issues and their solutions when using tmux with SSH sessions.

12.1 Session Not Found After Disconnect

Problem: After reconnecting to a server, tmux ls shows no sessions.

Cause: The tmux server process may have crashed or been killed.

Solution:

- Check if tmux server is running: ps aux | grep tmux

- Look for zombie sessions: ls -la /tmp/tmux-*/

- If using systemd, check: systemctl --user status tmux

- Prevent by using tmux-continuum for automatic saving

12.2 Colors Not Displaying Correctly

Problem: Applications like vim or htop show incorrect colors.

Solution: Set proper terminal type in ~/.tmux.conf:

```
set -g default-terminal "screen-256color"
set -ga terminal-overrides ",*256col*:Tc"
```

Also ensure your terminal emulator supports 256 colors and SSH is not overriding TERM.

12.3 Copy/Paste Not Working

Problem: Cannot copy text from tmux to system clipboard.

Solution: Install tmux-yank plugin or configure manually:

```
# For macOS with reattach-to-user-namespace
set -g default-command "reattach-to-user-namespace -l $SHELL"
# For Linux with xclip
bind C-c run "tmux save-buffer - | xclip -i -sel clipboard"
bind C-v run "tmux set-buffer $(xclip -o -sel clipboard); tmux paste-buffer"
```

12.4 SSH Connection Timeout

Problem: SSH connection drops even with tmux running.

Clarification: tmux does not prevent SSH disconnections; it preserves your work when they occur.

To reduce disconnections:

```
# Add to ~/.ssh/config on your local machine:
Host *
ServerAliveInterval 60
ServerAliveCountMax 3
# Or on the server in /etc/ssh/sshd_config:
ClientAliveInterval 60
ClientAliveCountMax 3
```

12.5 Cannot Attach to Session

Problem: "no sessions" error even though tmux ls shows sessions.

Cause: Permission issues or wrong socket path.

Solution:

```
# Check socket permissions
ls -la /tmp/tmux-$UID/
# Set correct TMUX_TMPDIR if needed
export TMUX_TMPDIR=/tmp
# Force attach with socket path
tmux -S /tmp/tmux-1000/default attach
```

12.6 Pane Layout Gets Messed Up

Problem: Pane sizes change unexpectedly when creating/destroying panes.

Solution: Use predefined layouts or lock pane sizes:

```
# Cycle through preset layouts: Ctrl+b Space
# Set specific pane dimensions
tmux split-window -h -p 30 # 30% width
tmux split-window -v -l 20 # 20 lines height
```

Summary

tmux is an essential tool for anyone who works with remote servers via SSH. Its client-server architecture solves the fundamental problem of fragile SSH connections by decoupling your terminal sessions from your network connection. By creating sessions that persist on the remote server, you can confidently run long-duration tasks knowing they will continue even if your connection drops, your laptop enters sleep mode, or you need to change locations.

The key practices for SSH session persistence with tmux are: (1) always create named sessions for important work, (2) detach properly using Ctrl+b d rather than closing the terminal, (3) install tmux-resurrect and tmux-continuum for automatic session saving, and (4) consider configuring SSH to auto-start tmux on connection.

With the knowledge from this guide, you can now work with remote servers confidently, knowing that your long-running tasks are protected from the vagaries of network connections. Whether you are compiling large projects, training machine learning models, deploying applications, or simply maintaining a persistent development environment, tmux provides the reliability and flexibility you need.