Rust Terminal Development Cheatsheet

Quick reference for terminal-based Rust development on Ubuntu.

Cargo Basics

Command	Description
`cargo new myproj`	Create new binary project
`cargo new mylib --lib`	Create new library project
`cargo build`	Build debug
`cargo build --release`	Build release
`cargo run`	Build and run
`cargo run --release`	Build and run release
`cargo check`	Fast syntax/type check (no codegen)
`cargo test`	Run tests
`cargo test -- --nocapture`	Tests with stdout
`cargo test test_name`	Run specific test
`cargo doc --open`	Generate and open docs
`cargo clean`	Remove target directory

`cargo update`	Update dependencies

Code Quality

Command	Description
`cargo fmt`	Format code
`cargo fmt --check`	Check formatting (CI)
`cargo clippy`	Run linter
`cargo clippy -- -D warnings`	Clippy with warnings as errors
`cargo clippy --fix`	Auto-fix lint issues
`cargo fix`	Auto-fix compiler warnings

Fast Feedback Loop

Command	Description
`cargo watch -c -x check`	Auto-check on save
`cargo watch -c -x test`	Auto-test on save
`cargo watch -c -x 'run -- args'`	Auto-run on save
`bacon`	TUI for cargo watch
`bacon test`	Bacon in test mode

`bacon clippy`	Bacon in clippy mode
`cargo nextest run`	Fast parallel test runner

Toolchain Management

Command	Description
`rustup update`	Update all toolchains
`rustup default stable`	Set default toolchain
`rustup toolchain list`	List installed toolchains
`rustup component add rust-src`	Add component
`rustup component list --installed`	List components
`cargo +nightly build`	Use nightly for one command
`rustup run nightly cargo build`	Alternative syntax
`rustc --print sysroot`	Print toolchain location
`rustup doc`	Open local Rust documentation

Compiler Inspection (Source → Binary Pipeline)

Macro Expansion

```
cargo expand                    # Full expansion
cargo expand main               # Expand main only
cargo expand path::to::module   # Expand specific module
```

HIR (High-level IR) [Nightly]

```
cargo +nightly rustc -- -Zunpretty=hir
cargo +nightly rustc -- -Zunpretty=hir,typed    # With type annotations
```

MIR (Mid-level IR) [Nightly]

```
cargo +nightly rustc -- -Zunpretty=mir          # Print MIR
cargo +nightly rustc -- -Zdump-mir=all          # Dump all MIR passes
cargo +nightly rustc -- -Zdump-mir=all -Zdump-mir-dir=./mir_dump
cargo +nightly rustc -- -Zmir-opt-level=0       # Unoptimized MIR
```

LLVM IR

```
# Via rustc
cargo rustc --release -- --emit=llvm-ir -C debuginfo=0 -C codegen-units=1
# Output: target/release/deps/*.ll

# Via cargo-show-asm
cargo asm --lib --llvm function_name
```

Assembly

```
# Via rustc (Intel syntax)
cargo rustc --release -- --emit=asm -C llvm-args=-x86-asm-syntax=intel
# Output: target/release/deps/*.s

# Via cargo-show-asm (recommended)
cargo asm --lib                     # List available functions
cargo asm --lib function_name       # Show assembly for function
cargo asm --lib --rust func_name    # Interleaved with Rust source
cargo asm --lib --mir func_name     # MIR via cargo-show-asm
```

Type and Memory Layout [Nightly]

```
RUSTFLAGS="-Zprint-type-sizes" cargo +nightly build 2>&1 | grep print-type-size
```

Binary Size Analysis

```
cargo bloat --release              # Function sizes
cargo bloat --release -n 20        # Top 20 functions
cargo bloat --release --crates     # Size by crate
```

All --emit Options

```
--emit=asm          # Assembly
--emit=llvm-ir      # LLVM IR
--emit=llvm-bc      # LLVM bitcode
--emit=mir          # MIR
--emit=obj          # Object file
--emit=link         # Final binary (default)
--emit=dep-info     # Dependency info
```

Debugging with rust-gdb

Starting a Session

```
# Build with debug symbols (default for dev builds)
cargo build

# Start debugger
rust-gdb target/debug/myprogram
rust-gdb --args target/debug/myprogram arg1 arg2
```

Essential GDB Commands

Command	Description
`r` / `run`	Run program
`c` / `continue`	Continue after breakpoint
`n` / `next`	Step over (next line)
`s` / `step`	Step into function

`finish`	Run until function returns
`q` / `quit`	Exit GDB

Breakpoints

Command	Description
`b main`	Break at main
`b main.rs:42`	Break at file:line
`b module::function`	Break at function
`b 42`	Break at line in current file
`info breakpoints`	List breakpoints
`delete 1`	Delete breakpoint #1
`disable 1`	Disable breakpoint
`enable 1`	Enable breakpoint

Inspecting State

Command	Description
`bt` / `backtrace`	Show call stack
`bt full`	Backtrace with locals
`frame 3`	Select stack frame
`info locals`	Show local variables

`info args`	Show function arguments
`p variable`	Print variable
`p *pointer`	Dereference pointer
`p struct.field`	Print struct field
`ptype variable`	Print variable type

Memory Inspection

Command	Description
`x/10x $sp`	10 hex words from stack
`x/s ptr`	Print as string
`x/i $pc`	Disassemble at PC
`info registers`	Show registers

TUI Mode

Command	Description
`layout src`	Show source window
`layout asm`	Show assembly window
`layout split`	Source + assembly
`layout regs`	Show registers
`tui disable`	Exit TUI mode

`Ctrl-x a`	Toggle TUI mode

Binary Inspection (Post-Compilation)

objdump

```
objdump -d target/debug/myprogram | less         # Disassemble
objdump --demangle -d target/debug/myprogram     # With demangled names
objdump -t target/debug/myprogram                # Symbol table
objdump -h target/debug/myprogram                # Section headers
```

readelf

```
readelf -h target/debug/myprogram    # ELF header
readelf -S target/debug/myprogram    # Section headers
readelf -s target/debug/myprogram    # Symbol table
readelf -d target/debug/myprogram    # Dynamic section
readelf -l target/debug/myprogram    # Program headers
```

nm (Symbol Table)

```
nm target/debug/myprogram                    # All symbols
nm --demangle target/debug/myprogram         # Demangled
nm --demangle --size-sort target/debug/myprogram  # Sorted by size
nm -C target/debug/myprogram | grep my_func  # Find function
```

size

```
size target/debug/myprogram          # Section sizes summary
size -A target/debug/myprogram       # Detailed breakdown
```

cargo-binutils (Cross-Platform)

```
cargo nm --release                   # nm via cargo
cargo objdump --release -- -d        # objdump via cargo
cargo size --release                 # size via cargo
```

Standard Library Source

Location

```
# Print std source path
rustc --print sysroot
# Full path: $(rustc --print sysroot)/lib/rustlib/src/rust/library/

# Directory structure:
# library/
# ├── core/     # #![no_std] basics
# ├── alloc/    # Vec, Box, String
# └── std/      # Full standard library
```

Searching

```
# Search with ripgrep
rg "fn drop" --type rust $(rustc --print sysroot)/lib/rustlib/src/rust/library/

# Search specific crate
rg "pattern" $(rustc --print sysroot)/lib/rustlib/src/rust/library/alloc/
```

CI/CD Tools

Command	Description
`cargo audit`	Check for vulnerabilities
`cargo deny check`	Check dependency policies
`cargo tarpaulin`	Generate test coverage
`cargo tarpaulin --out Html`	HTML coverage report

Environment Variables

Variable	Description
`RUST_BACKTRACE=1`	Enable backtraces
`RUST_BACKTRACE=full`	Full backtraces
`RUST_LOG=debug`	Log level (with `env_logger`)
`RUSTFLAGS="-Z..."`	Pass flags to rustc
`CARGO_INCREMENTAL=0`	Disable incremental compilation

Vim Quick Reference

Key	Action
`\b`	`:make build`
`\r`	`:make run`
`\t`	`:make test`
`\c`	`:make check`
`]q`	Next quickfix error
`[q`	Previous quickfix error
`:copen`	Open quickfix window
`:cclose`	Close quickfix window

Shell Aliases (from setup script)

```
# Basic
cb      # cargo build
cbr     # cargo build --release
cc      # cargo check
cr      # cargo run
ct      # cargo test
ctn     # cargo nextest run

# Watch
cw      # cargo watch -c -x check
cwt     # cargo watch -c -x test

# Inspection
mir     # View MIR
hir     # View HIR
casm    # cargo asm --lib
casmr   # cargo asm --lib --rust

# Functions
mir_dump [dir]     # Dump all MIR passes
llvm_ir            # Emit and view LLVM IR
emit_asm           # Emit and view assembly
std_search "query" # Search std library source
rust_stats         # Show environment info
```

Project Structure Best Practices

```
myproject/
├── .cargo/
│   └── config.toml       # Cargo configuration
├── src/
│   ├── main.rs           # Thin entry point
│   ├── lib.rs            # Application logic
│   └── prelude.rs        # Common re-exports
├── tests/                # Integration tests
├── benches/              # Benchmarks
├── Cargo.toml
├── Cargo.lock
├── rust-toolchain.toml   # Pinned toolchain
├── rustfmt.toml          # Formatting rules
├── clippy.toml           # Linter config
└── README.md
```

Quick Debugging Workflow

```
# 1. Check for errors fast
cargo check

# 2. Run tests
cargo nextest run

# 3. If crash/panic, get backtrace
RUST_BACKTRACE=1 cargo run

# 4. Debug specific issue
cargo build && rust-gdb target/debug/myprogram
(gdb) b main.rs:42
(gdb) r
(gdb) bt
(gdb) info locals

# 5. Inspect generated code
cargo asm --lib --rust problematic_function
```

Quick Inspection Workflow

```
# See what macros generate
cargo expand my_module

# See the control flow (MIR)
cargo +nightly rustc -- -Zunpretty=mir 2>&1 | less

# See the assembly
cargo asm --lib --rust my_function

# Check type sizes
RUSTFLAGS="-Zprint-type-sizes" cargo +nightly build 2>&1 | grep MyType
```