# NvChad (Italian version)

**A book from the Documentation Team**

**Version : 2024/10/27**

*Rocky Documentation Team*

# Table of contents

# 1. Licenza

RockyLinux offre corsi di Linux per formatori o persone che desiderano imparare ad amministrare un sistema Linux da soli.

I materiali di RockyLinux sono pubblicati ai sensi della licenza Creative Commons-BY-SA. Ciò significa che siete liberi di condividere e trasformare il materiale, rispettando i diritti dell'autore.

**BY** : **Attribution**. È necessario citare il nome dell'autore originale.

**SA** : **Share Alike**.

- Licenza Creative Commons-BY-SA: https://creativecommons.org/licenses/by-sa/4.0/

I documenti e le loro fonti sono liberamente scaricabili dal sito:

- https://docs.rockylinux.org
- https://github.com/rocky-linux/documentation

Le nostre fonti editoriali sono ospitate su github.com. Troverete il repository del codice sorgente dal quale è stata creata la versione di questo documento.

Da queste fonti è possibile generare il proprio materiale formativo personalizzato usando mkdocs. Troverete le istruzioni per generare il vostro documento [qui]. (https://github.com/rocky-linux/documentation/tree/main/build_pdf).

### Come posso contribuire al progetto di documentazione?

Troverete tutte le informazioni necessarie per unirvi a noi sulla nostra [home page del progetto git].(https://github.com/rocky-linux/documentation).

Auguriamo a tutti voi una piacevole lettura e speriamo che il contenuto sia di vostro gradimento.

# 2. Introduzione

In questo libro troverete il modo di implementare Neovim, insieme a NvChad, per creare un **ambiente** di **sviluppo integrato** (IDE) completamente funzionale.

Dico "modi" perché ci sono molte possibilità. L'autore si concentra sull'uso di questi strumenti per la scrittura di markdown, ma se il markdown non è il vostro obiettivo, non preoccupatevi, continuate a leggere. Se non conoscete nessuno di questi strumenti (NvChad o Neovim), questo libro vi fornirà un'introduzione a entrambi e, se seguirete questi documenti, vi renderete presto conto che potete configurare questo ambiente in modo che sia di grande aiuto per qualsiasi esigenza di programmazione o di scrittura di script.

Volete un IDE che vi aiuti a scrivere i playbook di Ansible? Puoi ottenerlo! Volete un IDE per Golang? Anche questo è disponibile. Volete semplicemente una buona interfaccia per scrivere script BASH? È anche disponibile.

Grazie all'uso dei **L**anguage **S**erver **P**rotocols e dei linters, è possibile configurare un ambiente personalizzato per l'utente. La cosa migliore è che una volta configurato l'ambiente, è possibile aggiornarlo rapidamente quando sono disponibili nuove modifiche attraverso l'uso di lazy.nvim e Mason, entrambi trattati qui.

Poiché Neovim è un fork di Vim, l'interfaccia generale sarà familiare agli utenti di *vim*. Se non siete utenti di *vim*, con questo libro imparerete rapidamente la sintassi dei comandi. Le informazioni trattate sono molte, ma è facile seguirle e, una volta completato il contenuto, saprete abbastanza per costruire il vostro IDE per *le vostre* esigenze con questi strumenti.

L'intento dell'autore era quello di **non** suddividere il libro in capitoli. Il motivo è che ciò implica un ordine da seguire che, nella maggior parte dei casi, non è necessario. *Si* consiglia di iniziare da questa pagina, leggere e seguire le sezioni "Software aggiuntivo", "Installare Neovim" e "Installare NvChad", ma da lì si può scegliere come procedere.

## 2.1 Utilizzo di Neovim come IDE

L'installazione di base di Neovim fornisce un ottimo editor per lo sviluppo, ma non può ancora essere definito un IDE; tutte le funzionalità IDE più avanzate, anche se già preimpostate, non sono ancora attivate. Per farlo, dobbiamo passare le configurazioni necessarie a Neovim, ed è qui che NvChad ci viene in aiuto. Questo ci permette di avere una configurazione di base con un solo comando!

La configurazione è scritta in Lua, un linguaggio di programmazione molto veloce che consente a NvChad di avere tempi di avvio e di esecuzione dei comandi e dei tasti molto rapidi. Ciò è reso possibile anche dalla tecnica di Lazy loading utilizzata per i plugin, che li carica solo quando necessario.

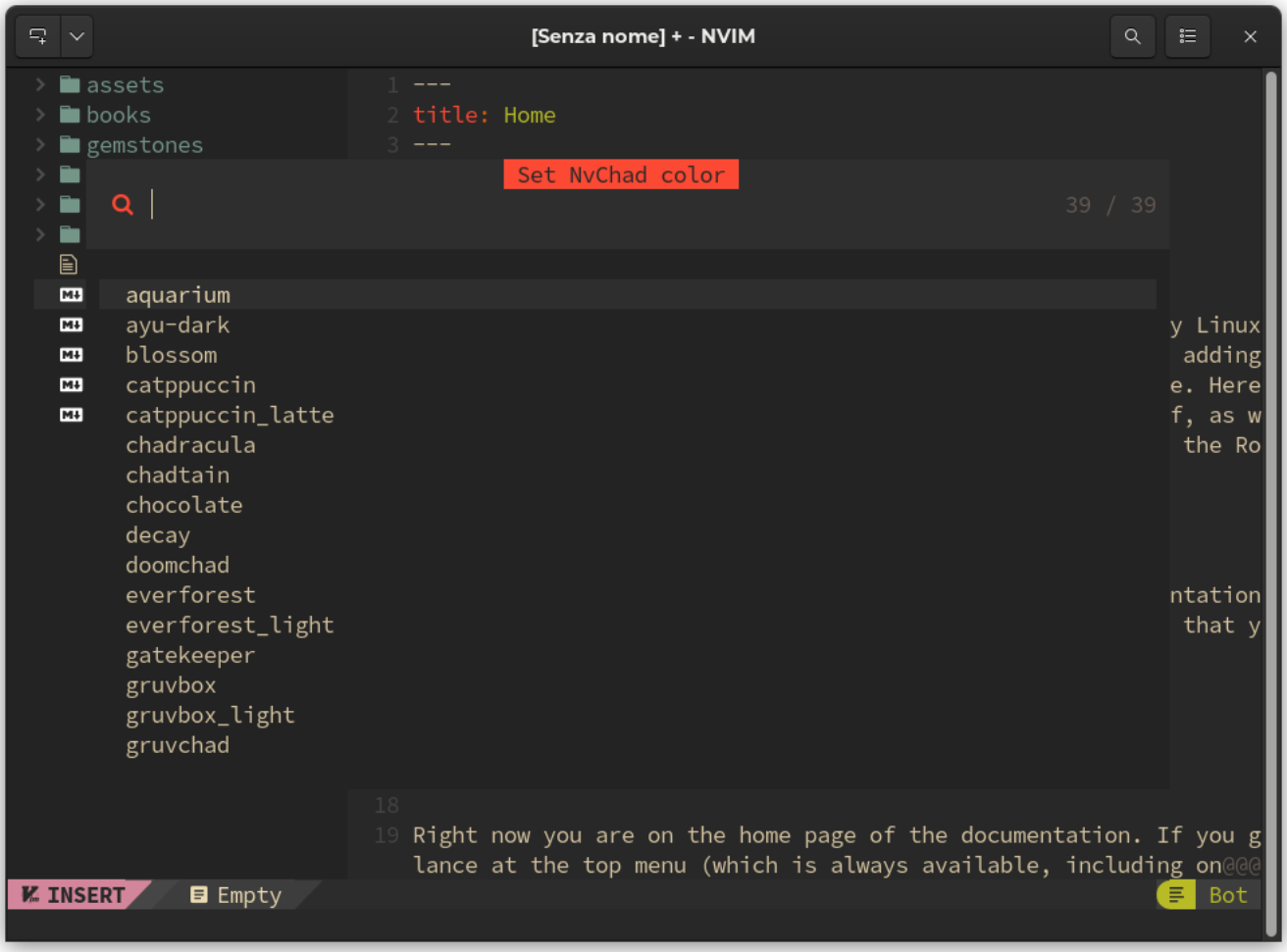
L'interfaccia risulta essere molto pulita e piacevole.

Come gli sviluppatori di NvChad tengono a precisare, il progetto vuole essere solo una base su cui costruire il proprio IDE personale. La successiva personalizzazione avviene attraverso l'uso di plugin.


## 2.1.1 Caratteristiche Principali

- **Progettato per essere veloce.** Dalla scelta del linguaggio di programmazione alle tecniche di caricamento dei componenti, tutto è stato pensato per ridurre al minimo i tempi di esecuzione.
- **Interfaccia attraente.** Nonostante sia un'applicazione *cli*, l'interfaccia ha un aspetto moderno e bello dal punto di vista grafico, inoltre tutti i componenti si adattano perfettamente all'interfaccia utente.
- **Estremamente configurabile.** Grazie alla modularità derivata dall'applicazione di base (NeoVim), l'editor può essere adattato perfettamente alle proprie esigenze. Tuttavia, ricordate che quando parliamo di personalizzazione, ci riferiamo alla funzionalità e non all'aspetto dell'interfaccia.

- **Meccanismo di aggiornamento automatico.** L'editor è dotato di un meccanismo (attraverso l'uso di *git*) che consente di effettuare aggiornamenti con un semplice comando `:NvChadUpdate`.
- **Powered by Lua**. La configurazione di NvChad è scritta interamente in *lua*, il che le permette di integrarsi perfettamente nella configurazione di Neovim, sfruttando tutte le potenzialità dell'editor su cui si basa.
- **Numerosi temi integrati.** La configurazione comprende già un gran numero di temi da utilizzare, ricordando sempre che stiamo parlando di un'applicazione *cli*, i temi possono essere selezionati con il tasto `<leader> + th`.



## 2.2 Riferimenti

### 2.2.1 Lua

#### Che cos’è Lua?

Lua è un linguaggio di scripting robusto e leggero che supporta diversi metodi di programmazione. Il nome "Lua" deriva dalla parola portoghese che significa "luna"

Lua è stato sviluppato all'Università Cattolica di Rio de Janeiro da Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes. Lo sviluppo è stato necessario perché fino al 1992 il Brasile era soggetto a rigide norme di importazione per hardware e software, quindi per pura necessità i tre programmatori hanno sviluppato il proprio linguaggio di scripting chiamato Lua.

Poiché Lua si concentra principalmente sugli script, è raramente utilizzato come linguaggio di programmazione autonomo. È invece più spesso utilizzato come linguaggio di scripting che può essere integrato (embedded) in altri programmi.

Lua è utilizzato nello sviluppo di videogiochi e motori di gioco (Roblox, Warframe...), come linguaggio di programmazione in molti programmi di rete (Nmap, ModSecurity...) e come linguaggio di programmazione in programmi industriali. Lua viene utilizzato anche come libreria che gli sviluppatori possono integrare nei loro programmi per abilitare le funzionalità di scripting agendo esclusivamente come parte integrante dell'applicazione host.

#### Come funziona Lua

Ci sono due componenti principali di Lua:

- L'interprete Lua
- La macchina virtuale Lua (VM)

Lua non viene interpretato direttamente attraverso un file Lua come altri linguaggi, ad esempio Python. Utilizza invece l'interprete Lua per compilare un file Lua in bytecode. L'interprete Lua è altamente portatile e in grado di funzionare su una moltitudine di dispositivi.

### Caratteristiche Principali

- Velocità: Lua è considerato uno dei linguaggi di programmazione più veloci tra i linguaggi di scripting interpretati; può eseguire compiti molto impegnativi dal punto di vista delle prestazioni più velocemente della maggior parte degli altri linguaggi di programmazione.
- Dimensioni: Lua ha dimensioni ridotte rispetto ad altri linguaggi di programmazione. Le dimensioni ridotte sono ideali per integrare Lua in più piattaforme, dai dispositivi embedded ai motori di gioco.
- Portabilità e integrazione: La portabilità di Lua è quasi illimitata. Qualsiasi piattaforma che supporti il compilatore C standard può eseguire Lua senza problemi. Lua non richiede complesse riscritture per essere compatibile con altri linguaggi di programmazione.
- Semplicità: Lua ha un design semplice ma fornisce potenti funzionalità. Una delle caratteristiche principali di Lua sono i meta-meccanismi, che consentono agli sviluppatori di implementare le proprie funzionalità. La sintassi è semplice e facilmente comprensibile, in modo che chiunque possa imparare Lua e utilizzarlo nei propri programmi.
- Licenza: Lua è un software libero e open-source distribuito sotto la licenza MIT. Questo permette a chiunque di utilizzarlo per qualsiasi scopo senza pagare alcuna licenza o royalty.

## 2.2.2 Neovim

Neovim è descritto in dettaglio nella sua pagina dedicata, quindi ci soffermeremo solo sulle caratteristiche principali, che sono:

- Prestazioni: Molto veloce.
- Personalizzabile: Ampio ecosistema di plugin e temi.
- Evidenziazione della sintassi: Integrazione con Treesitter e LSP (richiede alcune configurazioni aggiuntive).
- Multpiattaforma: Linux, Windows e macOS
- Licenza: Mit: Una licenza permissiva breve e semplice con condizioni che richiedono solo la conservazione del copyright e degli avvisi di licenza.

## 2.2.3 LSP

Che cos'è il **L**anguage **S**erver **P**rotocol?

Un server di linguaggio è una libreria di linguaggio standardizzata che utilizza una propria procedura (protocollo) per fornire il supporto a funzioni quali il completamento automatico, la definizione di goto o le definizioni di mouseover.

L'idea alla base del Language Server Protocol (LSP) è quella di standardizzare il protocollo di comunicazione tra strumenti e server, in modo che un singolo server linguistico possa essere riutilizzato in più strumenti di sviluppo. In questo modo, gli sviluppatori possono semplicemente integrare queste librerie nei loro editor e fare riferimento alle infrastrutture linguistiche esistenti, invece di personalizzare il loro codice per includerle.

## 2.2.4 tree-sitter

Tree-sitter consiste fondamentalmente in due componenti: un generatore di parser e una libreria di parsing incrementale. Può costruire un albero sintattico del file sorgente e aggiornarlo in modo efficiente a ogni modifica.

Un parser è un componente che scompone i dati in elementi più piccoli per facilitarne la traduzione in un'altra lingua o, come nel nostro caso, per passarli alla libreria di parsing. Una volta scomposto il file sorgente, la libreria di parsing analizza il codice e lo trasforma in un albero sintattico, consentendo di manipolare la struttura del codice in modo più intelligente. In questo modo è possibile migliorare (e velocizzare)

- evidenziazione della sintassi
- navigazione del codice
- refactoring
- oggetti e movimenti del testo

**LSP e complementarità tree-sitter**

Sebbene possa sembrare che i due servizi (LSP e tree-sitter) siano ridondanti, in realtà sono complementari in quanto LSP lavora a livello di progetto mentre tree-sitter lavora solo sul file sorgente aperto.

Ora che abbiamo illustrato un po' le tecnologie utilizzate per creare l'IDE, possiamo passare al software aggiuntivo necessario per configurare il nostro NvChad.

# 3. Software aggiuntivo richiesto

Esistono diversi software aggiuntivi che, pur non essendo necessari, facilitano l'uso complessivo di NvChad. Le sezioni seguenti illustrano il software e il suo utilizzo.

## 3.1 RipGrep

`ripgrep` è uno strumento di ricerca orientato alla riga di comando che ricerca ricorsivamente la directory corrente per un modello di *regex* (espressione regolare). Per impostazione predefinita, *ripgrep* rispetta le regole di *gitignore* e salta automaticamente i file/directory e i file binari nascosti. Ripgrep offre un

eccellente supporto su Windows, macOS e Linux, con binari disponibili per ogni release.

#### Installare RipGrep da EPEL

In entrambi Rocky Linux 8 e 9, è possibile installare RipGrep dall'EPEL. Per farlo, installare la `epel-release`, aggiornare il sistema e quindi installare `ripgrep`:

```
sudo dnf install -y epel-release
sudo dnf upgrade
sudo dnf install ripgrep
```

#### Installare RipGrep usando cargo

Ripgrep è un software scritto in *Rust* ed è installabile con l'utilità `cargo`. Da notare, tuttavia, che `cargo' non è installato dall'installazione predefinita di *rust*, quindi è necessario installarlo esplicitamente. Se si verificano errori con questo metodo, tornare all'installazione da EPEL.

```
dnf install rust cargo
```

Una volta installato il software necessario, si può installare `ripgrep` con:

```
cargo install ripgrep
```

L'installazione salverà l'eseguibile `rg` nella cartella `~/.cargo/bin`, che è al di fuori del PATH; per utilizzarlo a livello utente, sarà necessario collegarlo a `~/.local/bin/`.

```
ln -s ~/.cargo/bin/rg ~/.local/bin/
```

## 3.2 Verificare RipGrep

A questo punto si può verificare che tutto sia a posto con:

```
rg --version
ripgrep 13.0.0
```

```
-SIMD -AVX (compiled)
+SIMD +AVX (runtime)
```

RipGrep è necessario per le ricerche ricorsive con `:Telescope`.

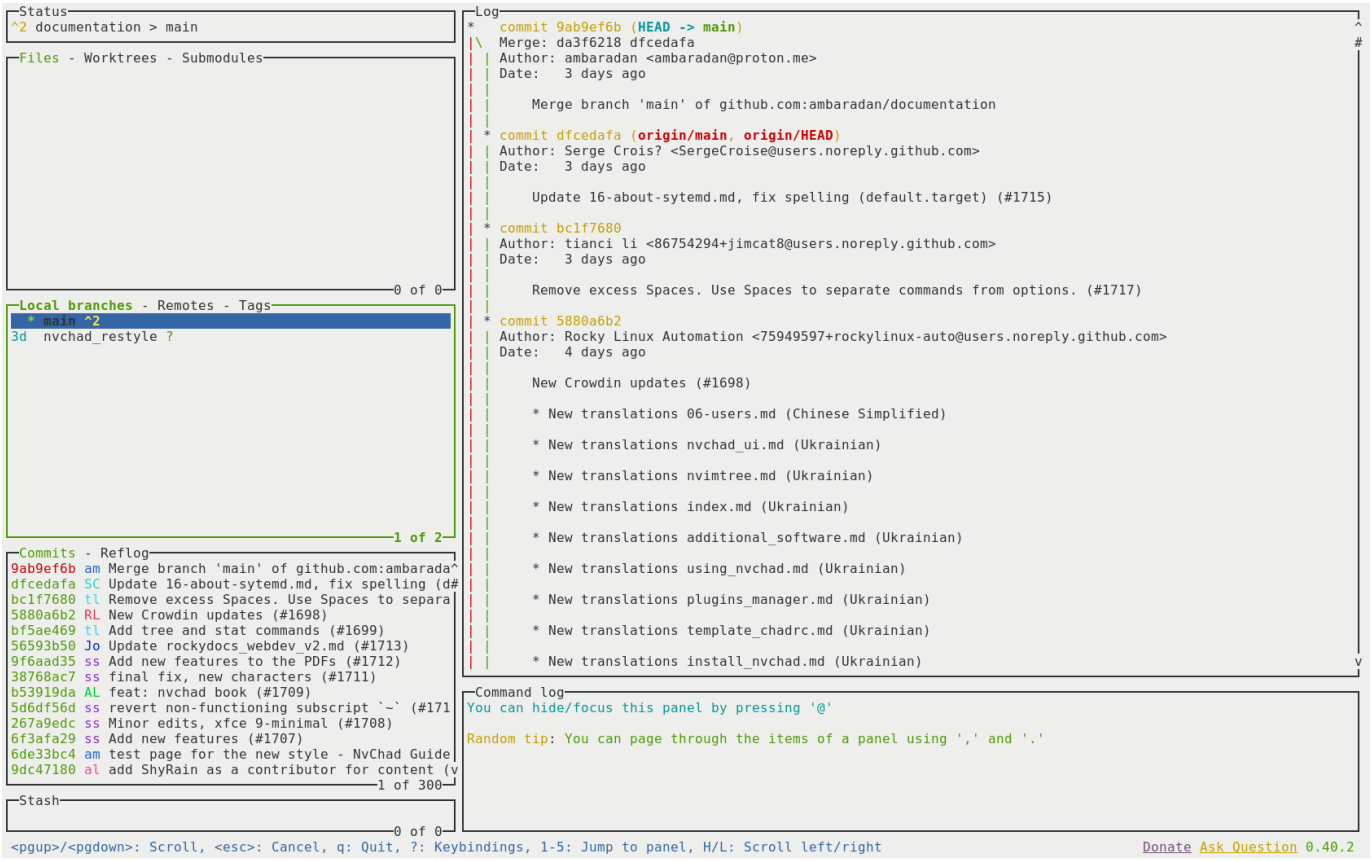
## 3.3 Lazygit

LazyGit è un'interfaccia in stile ncurses che consente di eseguire tutte le operazioni di `git` in maniera più agevole. È richiesto dal plugin lazygit.nvim. Questo plugin consente di utilizzare LazyGit direttamente da NvChad, aprendo una finestra fluttuante da cui è possibile eseguire tutte le operazioni sui repository, consentendo così di apportare tutte le modifiche al *repository di Git* senza uscire dall'editor.

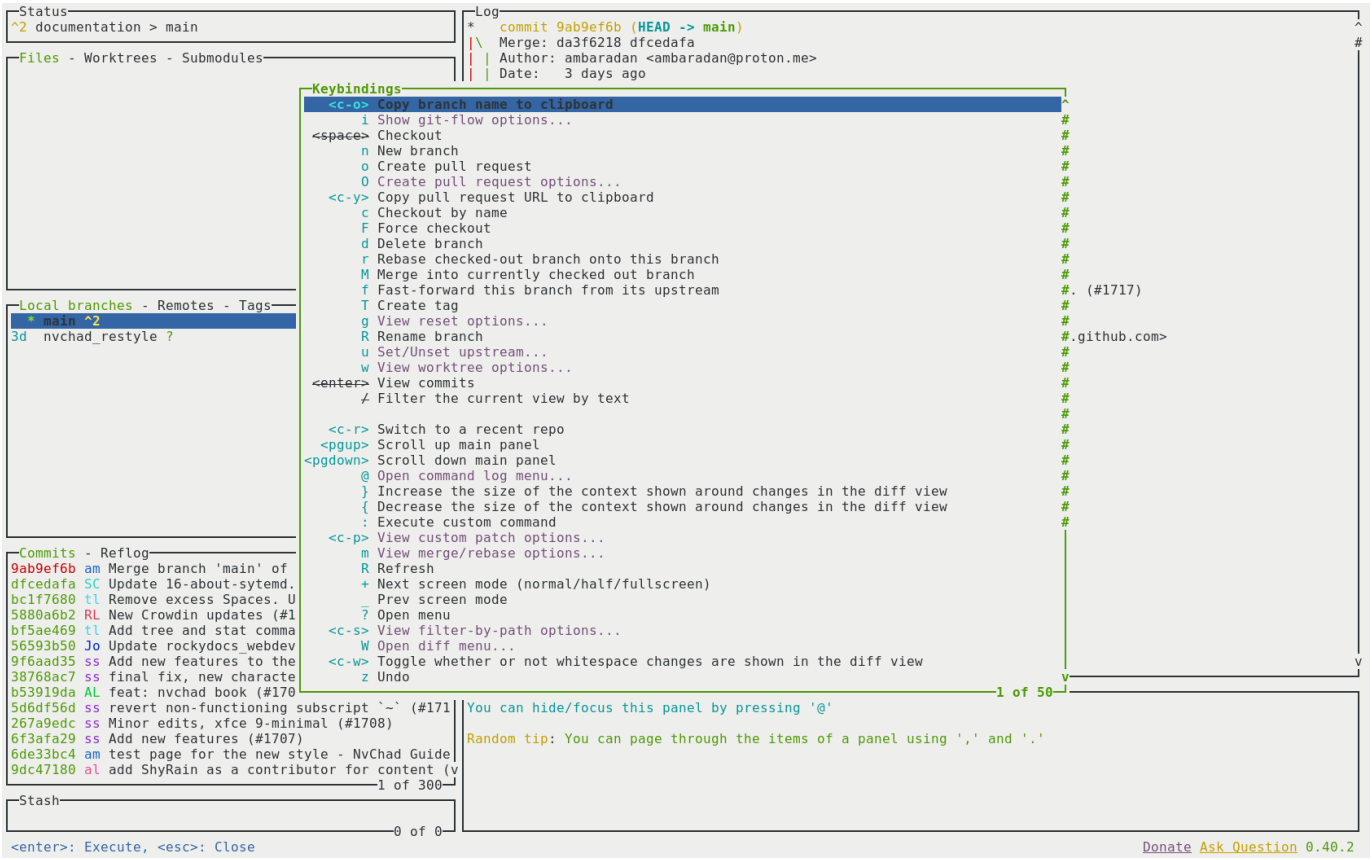
Per installarlo si può utilizzare il repository di Fedora. Su Rocky Linux 9 funziona perfettamente.

```
sudo dnf copr enable atim/lazygit -y
sudo dnf install lazygit
```

Una volta installato, aprire un terminale e digitare il comando `lazygit` e apparirà un'interfaccia simile a questa:




Con il tasto ? è possibile richiamare il menu con tutti i comandi disponibili.



Ora che tutti i software di supporto necessari sono presenti sul sistema, possiamo passare all'installazione del software di base. Inizieremo con l'editor su cui si basa l'intera configurazione, Neovim.

# 4. Installazione di Neovim

## 4.1 Introduzione a Neovim

Neovim è uno dei migliori editor di codice per la sua velocità, facilità di personalizzazione e configurazione.

Neovim è un fork dell'editor Vim. È nato nel 2014, principalmente a causa della mancanza, all'epoca, del supporto per i lavori asincroni in Vim. Scritto in linguaggio Lua con l'obiettivo di modularizzare il codice per renderlo più gestibile, Neovim è stato progettato pensando all'utente moderno. Come si legge sul sito ufficiale

> Neovim è stato creato per gli utenti che desiderano le parti migliori di Vim, e non solo.

Gli sviluppatori di Neovim hanno scelto Lua in quanto perfetto per l'incorporazione, l'utilizzo rapido di LuaJIT e una sintassi semplice e orientata agli script.

Dalla versione 0.5 Neovim include Treesitter (uno strumento per la generazione di parser) e supporta il Language Server Protocol (LSP). Questo riduce il numero di plugin necessari per ottenere funzioni di editing avanzate. Migliora le prestazioni di operazioni come il completamento del codice e il linting.

Uno dei suoi punti di forza è la personalizzazione. Tutte le sue configurazioni sono contenute in un unico file che può essere distribuito alle varie installazioni attraverso sistemi di controllo di versione (Git o altro) in modo che siano sempre sincronizzate.

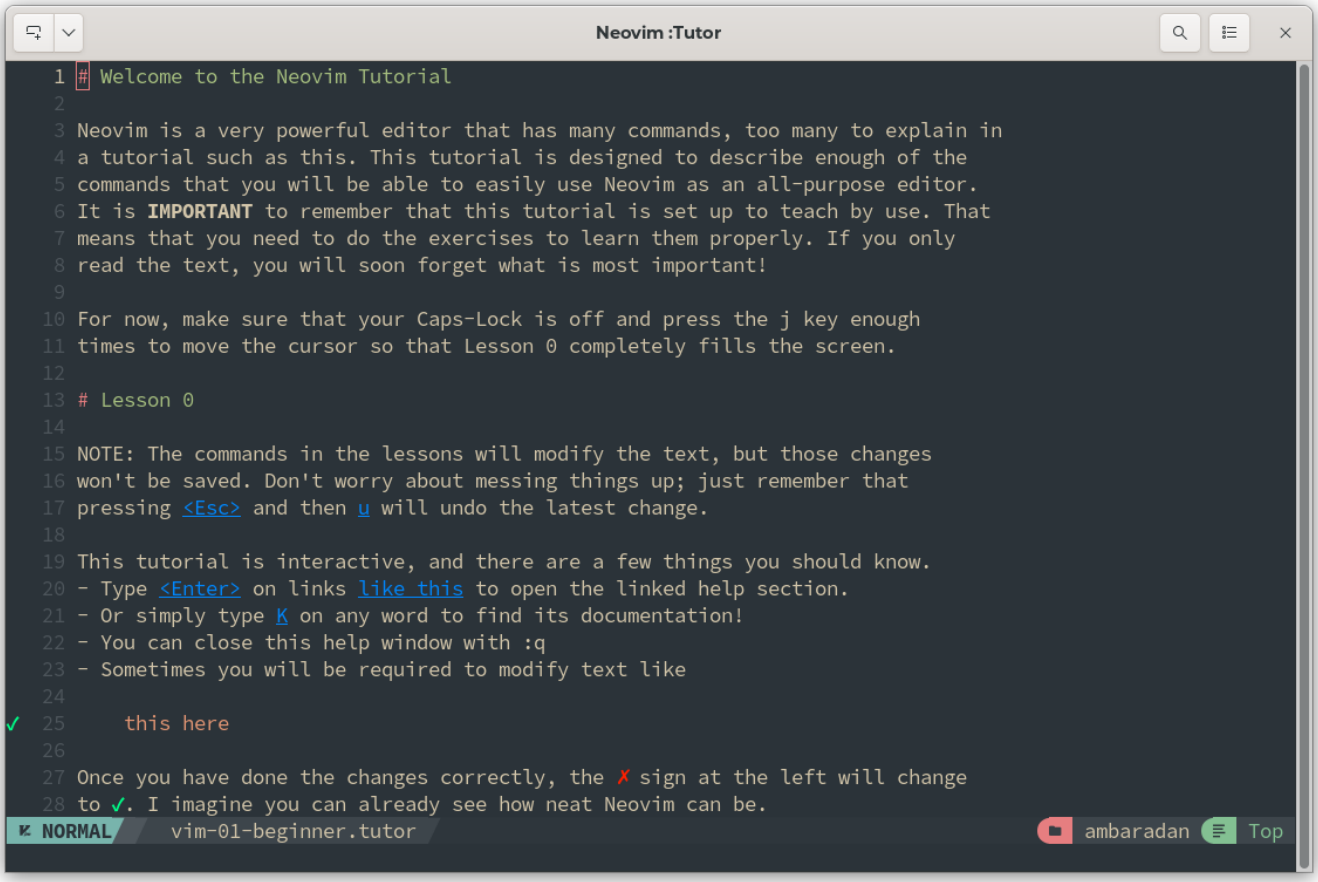
### 4.1.1 Comunità di sviluppatori

Sebbene Vim e Neovim siano entrambi progetti open-source e ospitati su GitHub, esiste una differenza significativa tra le modalità di sviluppo. Neovim ha uno sviluppo comunitario più aperto, mentre lo sviluppo di Vim è più legato alle scelte del suo creatore. La base di utenti e sviluppatori di Neovim è piuttosto piccola rispetto a Vim, ma è un progetto in continua crescita.

## 4.1.2 Caratteristiche Principali

- Prestazioni: Molto veloce.
- Personalizzabile: Ampio ecosistema di plugin e temi
- Evidenziazione della sintassi: Integrato con Treesitter e LSP, ma richiede qualche configurazione

Come per Vim, Neovim richiede una conoscenza di base dei suoi comandi e delle sue opzioni. È possibile ottenere una panoramica delle sue caratteristiche attraverso il comando `:Tutor` che richiama un file in cui è possibile leggerlo ed esercitarsi a usarlo. L'apprendimento richiede più tempo rispetto a un IDE completamente grafico, ma una volta imparate le scorciatoie per i comandi e le funzionalità incluse, si procederà in modo molto fluido nella modifica dei documenti.



## 4.2 Installazione di Neovim

**Installazione da EPEL**

Neovim è installabile anche dal repository EPEL. La versione disponibile è sempre troppo datata per soddisfare i requisiti minimi dell'installazione di NvChad.
L'installazione con questo metodo è fortemente sconsigliata e non è supportata da questa guida.

### Installazione da Pacchetto Precompilato

L'uso del pacchetto precompilato consente di installare sia la versione di sviluppo che quella stabile, che soddisfano i requisiti e possono essere utilizzate come base per la configurazione di NvChad.

Per utilizzare tutte le funzionalità dell'editor, è necessario soddisfare le dipendenze richieste da Neovim fornendo manualmente le dipendenze del pacchetto precompilato. I pacchetti necessari possono essere installati con:

```
dnf install compat-lua-libs libtermkey libtree-sitter libvterm luajit
luajit2.1-luv msgpack unibilium xsel
```

Dopo aver installato le dipendenze necessarie, è il momento di acquisire il pacchetto scelto.

Accedendo alla pagina delle release sarà possibile scaricare la versione di sviluppo (pre-release) o la versione stabile (stable). In entrambi i casi l'archivio compresso da scaricare per la nostra architettura è linux64.

Il file richiesto è nvim-linux64.tar.gz, è necessario scaricare anche il file nvim-linux64.tar.gz.sha256sum per verificarne l'integrità.

Supponendo che entrambi siano stati scaricati nella stessa cartella, utilizzeremo il seguente comando per la verifica:

```
sha256sum -c nvim-linux64.tar.gz.sha256sum
nvim-linux64.tar.gz: OK
```

Ora scompattate il pacchetto precompilato in una posizione all'interno della vostra cartella home; in questa guida è stata scelta la posizione `.local/share/`, ma può essere modificata in base alle proprie esigenze. Eseguire il comando:

```
tar xvzf nvim-linux64.tar.gz -C ~/.local/share/
```

A questo punto non rimane che creare un collegamento simbolico in ~/.local/bin/ per l'eseguibile nvim del pacchetto precompilato.

```
cd ~/.local/bin/
ln -sf ~/.local/share/nvim-linux64/bin/nvim nvim
```

Per verificare la corretta installazione, eseguire in un terminale il comando `nvim -v`, che dovrebbe ora mostrare qualcosa di simile: