Gestiona dotfiles fácilmente con GNU Stow

Edison Achalma
Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga

Resumen

This tutorial provides a step-by-step guide to managing dotfiles using GNU Stow, a tool that leverages symbolic links to centralize and synchronize configuration files across Unix-like systems (Linux, macOS, WSL). It explains the importance of dotfiles, such as .bashrc and .gitconfig, in customizing user environments and highlights the inefficiencies of manual management. The guide details installing GNU Stow, creating a dotfiles repository, linking configurations, and automating the process with a bash script. Advanced tips include handling conflicts, platform-specific setups, and alternatives like Chezmoi and YADM. This resource is designed for developers seeking efficient, portable configuration management.

Palabras Claves: Dotfiles, GNU Stow, Symbolic links, Configuration management, Git integration

Tabla de contenidos

Edison Achalma https://orcid.org/0000-0001-6996-3364

El autor no tiene conflictos de interés que revelar. Los roles de autor se clasificaron utilizando la taxonomía de roles de colaborador (CRediT; https://credit.niso.org/) de la siguiente manera: Edison Achalma: conceptualización, redacción

La correspondencia relativa a este artículo debe dirigirse a Edison Achalma, Email: elmer.achalma.09@unsch.edu.pe

Gestiona dotfiles fácilmente con GNU Stow

¿Alguna vez has perdido horas configurando tu terminal o editor tras cambiar de computadora? Los **dotfiles**, esos archivos ocultos como `.bashrc` o `.gitconfig`, guardan tus personalizaciones, pero gestionarlos a mano es un caos. **GNU Stow** simplifica todo: organiza tus configuraciones en un repositorio central y usa **enlaces simbólicos** para sincronizarlas en minutos.

¿Qué es Dotfiles?

Los **dotfiles** son archivos ocultos en sistemas Unix (Linux, macOS) que empiezan con un punto (ej., `.zshrc`, `.gitconfig`, `.config/nvim`). Almacenan configuraciones personalizadas para tu terminal, editor de código o gestor de ventanas. Por ejemplo, `.bashrc` define alias y variables de entorno, mientras que `.vimrc` ajusta tu editor Vim. Estos archivos son el corazón de tu flujo de trabajo, ya que personalizan tus herramientas favoritas.

Tener **dotfiles** bien organizados te ahorra horas al replicar tu entorno en nuevas máquinas. Imagina configurar tu shell o editor desde cero tras reinstalar tu sistema: ¡es tedioso! Con una gestión adecuada, puedes clonar tus configuraciones y tener todo listo rápidamente. Esto es importante para desarrolladores que trabajan en múltiples dispositivos o entornos como servidores y laptops.

Problemas de la Gestión Manual

Copiar **dotfiles** manualmente o usar scripts caseros es lento y arriesgado. Puedes sobrescribir archivos, olvidar configuraciones o perderlas en una reinstalación. Por ejemplo, mover `.zshrc` a otra máquina sin un sistema organizado puede causar errores si las versiones del software difieren. **GNU Stow** soluciona esto al centralizar tus archivos y crear **enlaces simbólicos** automáticamente, manteniendo todo sincronizado.

¿Qué es GNU Stow?

GNU Stow es un **gestor de granjas de enlaces simbólicos** (symlink farm manager) que permite administrar múltiples paquetes de software o conjuntos de archivos de configuración de manera organizada. Concepto principal:

```
Instalar cada paquete en su propio árbol de directorios
     ↓
Usar enlaces simbólicos para que aparezcan en un árbol común
     ↓
Administrar fácilmente cada paquete de forma independiente
```

Problema original:

```
# En /usr/local/man/man1 tenías:
a2p.1      # ¿De qué paquete es?
perl.1     # ¿Perl?
emacs.1    # ¿Emacs?
etags.1    # ¿Emacs también?

# Al desinstalar Perl... ¿qué archivos eliminar?
```

Solución con Stow:

```
# Cada paquete en su propio árbol:
/usr/local/stow/perl/
   bin/
       perl
       a2p
   man/
       man1/
           perl.1
           a2p.1

/usr/local/stow/emacs/
   bin/
       emacs
   man/
       man1/
           emacs.1

# Stow crea symlinks en /usr/local/ que apuntan a los paquetes
```

Gestión de Dotfiles

Aunque Stow fue diseñado para software, hoy en día su uso principal es **gestionar dotfiles**:

Ventajas:

- Mantener dotfiles organizados por aplicación
- Sincronizar con Git
- Instalar/desinstalar configuraciones selectivamente
- Mantener backups sin perder estructura
- Compartir configuraciones entre máquinas
- Control de versiones granular

Comparación: Antes vs Después de Stow
Sin Stow:

```
~/.config/
   nvim/
   kitty/
   zsh/
   ...

# Problemas:
# - Difícil hacer backup selectivo
# - No hay organización por paquete
# - Complicado compartir entre máquinas
# - Sin control de versiones granular
```

Con Stow:

```
~/dotfiles/          # Stow directory
   nvim/           # Package
       .config/
           nvim/
   kitty/          # Package
       .config/
           kitty/
   zsh/            # Package
       .zshrc
       .zshenv

# Ventajas:
# - Cada aplicación es un "paquete"
# - Fácil stow/unstow selectivo
# - Git maneja cada paquete independientemente
# - Estructura clara y mantenible
```

1 Instalación

1.1 Linux

Ubuntu/Debian:

```
sudo apt update
sudo apt install stow
```

Arch Linux:

```
sudo pacman -S stow
```

Fedora/RHEL:

```
sudo dnf install stow
```

openSUSE:

```
sudo zypper install stow
```

1.2 macOS

```
# Con Homebrew
brew install stow

# O con MacPorts
sudo port install stow
```

1.3 Desde Fuente

```
# Descargar última versión
wget https://ftp.gnu.org/gnu/stow/stow-latest.tar.gz
tar -xzf stow-latest.tar.gz
cd stow-2.4.1

# Compilar e instalar
./configure
make
sudo make install
```

1.4 Verificar Instalación

```
# Verificar versión
stow --version
# GNU Stow version 2.4.1

# Ver ayuda
stow --help
```

2 Conceptos Fundamentales

2.1 Terminología Clave

2.1.1 Package (Paquete)

Una colección relacionada de archivos y directorios que administras como una unidad.

```
# Ejemplo: paquete "nvim"
nvim/
   .config/
       nvim/
           init.lua
           lua/
   .local/
       share/
           nvim/
```

2.1.2 Target Directory (Directorio Objetivo)

El directorio raíz donde quieres que aparezcan instalados tus paquetes.

```
# Para dotfiles, usualmente es:
Target: ~/ (tu HOME)

# Para software del sistema:
Target: /usr/local
```

2.1.3 Stow Directory (Directorio Stow)

El directorio raíz que contiene todos tus paquetes en subdirectorios separados.

```
# Para dotfiles:
Stow dir: ~/dotfiles/

# Para software:
Stow dir: /usr/local/stow/
```

2.1.4 Installation Image (Imagen de Instalación)

La estructura de archivos y directorios requerida por un paquete, relativa al target directory.

```
# El paquete "zsh" tiene esta imagen:
zsh/
   .zshrc          # → ~/.zshrc
   .zshenv         # → ~/.zshenv
   .config/
       zsh/        # → ~/.config/zsh/
           aliases.zsh
```

2.1.5 Symlink (Enlace Simbólico)

Un archivo especial que apunta a otro archivo o directorio.

```
# Ejemplo:
~/.zshrc -> ~/dotfiles/zsh/.zshrc
         ↑
      symlink
```

Tipos de symlinks:

- **Absoluto:** `/home/user/dotfiles/zsh/.zshrc`
- **Relativo:** `../dotfiles/zsh/.zshrc`

Nota: Stow solo crea symlinks **relativos** dentro del target directory.

2.2 Jerarquía de Directorios

```
   /home/user/  (target directory)

   .zshrc
   .config/
        nvim/      symlinks
        kitty/


                   ↓
```

```
/home/user/dotfiles/  (stow dir)

   zsh/     (package)
       .zshrc
   nvim/    (package)
       .config/
           nvim/
   kitty/   (package)
       .config/
           kitty/
```

3 Sintaxis y Comandos

3.1 Sintaxis Básica

```
stow [opciones] [flags de acción] paquete1 paquete2 ...
```

3.2 Acciones Principales

3.2.1 Stow (Instalar)

```
# Instalar un paquete
stow nombre-paquete

# Instalar múltiples paquetes
stow nvim zsh kitty

# Flag explícito (opcional)
stow -S nvim
stow --stow nvim
```

3.2.2 Delete (Desinstalar)

```
# Desinstalar un paquete
stow -D nvim
stow --delete nvim

# Desinstalar múltiples
stow -D nvim zsh kitty
```

3.2.3 Restow (Reinstalar)

```
# Unstow + Stow en una operación
stow -R nvim
stow --restow nvim

# Útil después de actualizar paquete
```

3.3 Opciones de Directorio

3.3.1 `-d` / `--dir` (Stow Directory)

```
# Especificar stow directory
stow -d ~/mis-dotfiles -t ~ nvim

# Default: directorio actual
```

3.3.2 `-t` / `--target` (Target Directory)

```
# Especificar target directory
stow -t /usr/local perl

# Default: padre del stow directory
```

Ejemplo completo:

```
# Estructura:
/opt/
    myapps/         # stow directory
        myapp/      # package
            bin/
                myapp

# Comando:
cd /opt/myapps
stow -t /usr/local myapp

# Resultado:
/usr/local/bin/myapp -> ../opt/myapps/myapp/bin/myapp
```

3.4 Opciones de Simulación y Verbosidad

3.4.1 `-n` / `--no` / `--simulate` (Dry Run)

```
# Mostrar qué haría sin hacer cambios
stow -n nvim
stow --simulate nvim

# Combinado con verbose
stow -nv nvim
```

3.4.2 `-v` / `--verbose` (Verbosidad)

```
# Niveles de verbosidad: 0-5
stow -v nvim          # verbose level 1
stow -vv nvim         # verbose level 2
stow --verbose=5 nvim # verbose level 5

# Nivel 0: silencioso (default)
# Nivel 1-2: operaciones principales
# Nivel 3-5: debug detallado
```

Ejemplo:

```
$ stow -nv zsh
WARNING! stowing zsh would cause conflicts:
  * existing target is neither a link nor a directory: .zshrc
All operations aborted.
```

3.5 Opciones Avanzadas

3.5.1 `--ignore` (Ignorar Archivos)

```
# Ignorar archivos que coincidan con regexp
stow --ignore='.*\.orig' --ignore='.*\.dist' nvim

# Múltiples patrones
stow --ignore='README.*' --ignore='.*~' nvim
```

3.5.2 `--defer` (Diferir)

```
# No sobrescribir si ya existe desde otro paquete
stow --defer=man --defer=info perl
```

3.5.3 `--override` (Sobrescribir)

```
# Forzar sobrescribir symlinks existentes
stow --override=man --override=info perl
```

3.5.4 `--dotfiles` (Modo Dotfiles)

```
# Transforma "dot-" en "."
# dot-bashrc → .bashrc
stow --dotfiles bash

# Ejemplo de paquete:
bash/
    dot-bashrc    # Se convierte en ~/.bashrc
```

3.5.5 `--no-folding` (Sin Tree Folding)

```
# Desactivar tree folding
stow --no-folding nvim

# Crea directorios en lugar de symlinks a directorios
```

3.5.6 `--adopt` (Adoptar Archivos)

```
# CUIDADO: Modifica el stow directory
# Mueve archivos del target al package

stow --adopt nvim

# Si ~/.config/nvim/init.lua existe:
# Lo mueve a ~/dotfiles/nvim/.config/nvim/init.lua
# Luego crea el symlink
```

3.6 Combinando Operaciones

```
# Mezclar múltiples acciones
stow -D old-nvim -S new-nvim

# Orden de ejecución:
# 1. Unstow old-nvim
# 2. Stow new-nvim

# Múltiples paquetes, múltiples acciones
stow -S pkg1 pkg2 -D pkg3 pkg4 -S pkg5 -R pkg6
# Resultado: unstow pkg3,4,6 → stow pkg1,2,5,6
```

4 Estructura de Directorios

4.1 Estructura Recomendada para Dotfiles

```
~/dotfiles/                    # Stow directory
   git/                       # Package
       .gitconfig
   zsh/                       # Package
       .zshrc
       .zshenv
       .config/
           zsh/
               aliases.zsh
               functions.zsh
   nvim/                      # Package
```

```
      .config/
          nvim/
              init.lua
              lua/
                  plugins/
                  config/
  kitty/                      # Package
      .config/
          kitty/
              kitty.conf
              themes/
  tmux/                       # Package
      .tmux.conf
      .config/
          tmux/
  kde/                        # Package
       .config/
           kdeglobals
           dolphinrc
           kwinrc
```

4.2 Principios de Organización

4.2.1 Un Directorio = Un Paquete

```
# Bien: un paquete por aplicación
nvim/
     .config/
         nvim/

# Mal: múltiples aplicaciones en un paquete
editors/
     .config/
        nvim/
        vim/
     .vimrc
```

4.2.2 Replicar Estructura del HOME

```
# El contenido del paquete debe replicar la estructura de ~/

# Ejemplo: archivo en ~/.config/kitty/kitty.conf
# Paquete debe ser:
kitty/
     .config/              # replica la estructura
         kitty/
             kitty.conf
```

```
# NO:
kitty/
    kitty.conf            # falta .config/
```

4.2.3 Agrupar Lógicamente

```
# Opción 1: Por aplicación
~/dotfiles/
  nvim/
  vim/
  emacs/

# Opción 2: Por categoría (menos común)
~/dotfiles/
  editors/
      .vimrc
      .config/nvim/
  shells/
      .zshrc
      .bashrc

# Recomendado: Opción 1 (por aplicación)
```

4.3 Ejemplos de Estructuras

4.3.1 Estructura Simple

```
~/dotfiles/
  bash/
      .bashrc
  git/
      .gitconfig
  vim/
      .vimrc
```

Instalación:

```
mkdir ~/dotfiles
cd ~/dotfiles

# Crear la estructura para bash
mkdir -p bash

# o mueve, o crea symlink, como prefieras
cp ~/.bashrc bash/.bashrc
# (opcional) cp ~/.bash_profile bash/.bash_profile
```

```
# Para git
mkdir git
cp ~/.gitconfig git/.gitconfig

# Para vim
mkdir vim
cp ~/.vimrc vim/.vimrc
# si tienes ~/.vim/ con plugins, etc → también lo copias/mueves

# Para zsh + oh-my-zsh customizaciones
mkdir -p zsh/.config
cp ~/.zshrc zsh/.zshrc
```

Una vez que tengas (por ejemplo) la carpeta bash/ con .bashrc dentro:

```
cd ~/dotfiles

# Instalar un paquete
stow bash       # → crea symlink ~/.bashrc → ~/dotfiles/bash/.bashrc
stow git
stow vim

# Instalar múltiples paquetes
stow bash git vim nvim kitty zsh
```

Resultado:

```
~/.bashrc -> dotfiles/bash/.bashrc
~/.gitconfig -> dotfiles/git/.gitconfig
~/.vimrc -> dotfiles/vim/.vimrc
```

4.3.2 Estructura Compleja

```
~/dotfiles/
   shell/
       .bashrc
       .zshrc
       .config/
           bash/
               aliases.bash
           zsh/
               aliases.zsh
   terminal/
       .config/
           kitty/
               kitty.conf
               themes/
```

```
          alacritty/
              alacritty.yml
   editor/
       .config/
           nvim/
               init.lua
               lua/
                   plugins.lua
```

5 Instalación de Paquetes

5.1 Proceso de Instalación

5.1.1 Tree Folding (Plegado de Árbol)

Stow intenta crear el **mínimo número de symlinks** posible.
Ejemplo 1: Target Vacío

```
# Estado inicial:
~/ (vacío, sin ~/.config/)

# Paquete:
~/dotfiles/nvim/
     .config/
         nvim/
             init.lua

# Comando:
cd ~/dotfiles
stow nvim

# Resultado (tree folding):
~/.config -> dotfiles/nvim/.config/

# En lugar de:
# ~/.config/nvim/init.lua -> ...
# Stow crea un symlink al directorio completo
```

Ejemplo 2: Target con Archivos Existentes

```
# Estado inicial:
~/.config/
     kitty/        # ya existe
         kitty.conf

# Paquete:
~/dotfiles/nvim/
     .config/
         nvim/
```

```
            init.lua

# Comando:
stow nvim

# Resultado (NO puede hacer tree folding):
~/.config/              # directorio real
    kitty/            # ya existía
        kitty.conf
    nvim -> ../dotfiles/nvim/.config/nvim/
```

5.1.2 Tree Unfolding (Desplegado de Árbol)

Cuando un symlink plegado debe ser "abierto" para acomodar otro paquete.
Escenario:

```
# Estado inicial:
~/.config -> dotfiles/nvim/.config/

# Instalar otro paquete:
~/dotfiles/kitty/
    .config/
        kitty/
            kitty.conf

# Comando:
stow kitty

# Proceso de unfolding:
# 1. Eliminar symlink: ~/.config
# 2. Crear directorio: ~/.config/
# 3. Crear symlinks:
#    ~/.config/nvim -> ../dotfiles/nvim/.config/nvim/
#    ~/.config/kitty -> ../dotfiles/kitty/.config/kitty/
```

5.2 Instalación Básica

```
# Crea el directorio
mkdir ~/dotfiles

# Navegar al stow directory
cd ~/dotfiles

# Instalar un paquete
stow nvim

# Instalar múltiples paquetes
```

```
stow nvim zsh git kitty

# Instalar todos los paquetes
stow */
```

5.3 Instalación con Verificación

```
# Dry run primero (simular)
stow -nv nvim

# Si todo OK, instalar realmente
stow nvim

# Verificar symlinks creados
ls -la ~/.config/nvim
```

5.4 Instalación Selectiva

```
# Solo paquetes de terminal
stow kitty alacritty tmux

# Solo paquetes de shell
stow bash zsh fish

# Solo paquetes de editor
stow nvim vim emacs
```

6 Desinstalación de Paquetes

6.1 Proceso de Desinstalación

6.1.1 Eliminación de Symlinks

```
# Paquete instalado:
~/.zshrc -> dotfiles/zsh/.zshrc

# Desinstalar:
cd ~/dotfiles
stow -D zsh

# Resultado:
# ~/.zshrc eliminado (porque era symlink a stow package)
```

6.1.2 Eliminación de Directorios Vacíos

```
# Antes:
~/.config/
    nvim -> ../dotfiles/nvim/.config/nvim/

# Desinstalar:
stow -D nvim

# Después:
# ~/.config/ eliminado (si quedó vacío)
```

6.1.3 Tree Refolding (Re-plegado)

Después de eliminar symlinks, si un directorio contiene solo symlinks a un único paquete, Stow lo "re-pliega".

Escenario:

```
# Estado actual:
~/.config/
    nvim -> ../dotfiles/nvim/.config/nvim/
    kitty -> ../dotfiles/kitty/.config/kitty/

# Desinstalar kitty:
stow -D kitty

# Resultado (refolding):
~/.config -> dotfiles/nvim/.config/
```

6.2 Desinstalación Básica

```
# Navegar al stow directory
cd ~/dotfiles

# Desinstalar un paquete
stow -D nvim

# Desinstalar múltiples paquetes
stow -D nvim zsh git

# Desinstalar todos los paquetes
stow -D */
```

6.3 Desinstalación con Verificación

```
# Dry run primero
stow -Dnv nvim

# Si todo OK, desinstalar realmente
```

```
stow -D nvim

# Verificar que symlinks fueron eliminados
ls -la ~/.config/nvim
```

6.4 Desinstalación Parcial

```
# Desinstalar solo configuraciones de terminal
stow -D kitty alacritty tmux

# Mantener el resto
```

7 Reinstalación de Paquetes

7.1 Comando Restow

```
# Restow = Unstow + Stow
stow -R nvim
stow --restow nvim
```

7.2 Cuándo Usar Restow

1. Después de actualizar un paquete:

```
# Editaste archivos en ~/dotfiles/nvim/
cd ~/dotfiles
stow -R nvim

# Esto actualiza los symlinks si la estructura cambió
```

2. Para limpiar symlinks obsoletos:

```
# Eliminaste archivos del paquete
stow -R nvim

# Restow elimina symlinks huérfanos
```

3. Después de cambiar estructura:

```
# Moviste archivos dentro del paquete
# Antes: nvim/.vimrc
# Ahora: nvim/.config/nvim/init.lua

stow -R nvim
```

7.3 Restow vs Delete + Stow

```
# Método 1: Restow (recomendado)
stow -R nvim

# Método 2: Manual (equivalente)
stow -D nvim
stow nvim

# Ventaja de -R: más rápido, optimizado
```

8 Gestión de Dotfiles

8.1 Setup Inicial

8.1.1 Crear Estructura

```
# Crear directorio para dotfiles
mkdir -p ~/dotfiles
cd ~/dotfiles

# Inicializar Git
git init
```

8.1.2 Mover Configuraciones Existentes

Método manual:

```
# Crear paquete
mkdir -p ~/dotfiles/zsh

# Mover archivos
mv ~/.zshrc ~/dotfiles/zsh/
mv ~/.zshenv ~/dotfiles/zsh/

# Stow
cd ~/dotfiles
stow zsh
```

Con script:

```
#!/bin/bash
# migrate-to-stow.sh

DOTFILES="$HOME/dotfiles"
mkdir -p "$DOTFILES"

# Migrar zsh
mkdir -p "$DOTFILES/zsh"
mv ~/.zshrc "$DOTFILES/zsh/"
```

```
mv ~/.zshenv "$DOTFILES/zsh/"

# Migrar nvim
mkdir -p "$DOTFILES/nvim/.config"
mv ~/.config/nvim "$DOTFILES/nvim/.config/"

# Migrar git
mkdir -p "$DOTFILES/git"
mv ~/.gitconfig "$DOTFILES/git/"

# Stow todo
cd "$DOTFILES"
stow zsh nvim git
```

8.1.3 Usar `--adopt` (Con Precaución)

```
# Crear estructura primero
mkdir -p ~/dotfiles/nvim/.config
mkdir ~/dotfiles/nvim/.config/nvim

# Adoptar configuración existente
cd ~/dotfiles
stow --adopt nvim

# Esto MUEVE ~/.config/nvim/* a ~/dotfiles/nvim/.config/nvim/
# Y luego crea el symlink
```

8.2 Workflow Diario

8.2.1 Editar Configuraciones

```
# Los symlinks te permiten editar en cualquier lugar:

# Opción 1: Editar en home (a través del symlink)
nvim ~/.zshrc              # Edita ~/dotfiles/zsh/.zshrc

# Opción 2: Editar directamente en dotfiles
nvim ~/dotfiles/zsh/.zshrc # Mismo archivo
```

8.2.2 Agregar Nueva Aplicación

```
# 1. Crear paquete
cd ~/dotfiles
mkdir -p new-app/.config/new-app

# 2. Agregar archivos
```

```
cp -r ~/.config/new-app/* new-app/.config/new-app/

# 3. Remover originales
rm -rf ~/.config/new-app

# 4. Stow
stow new-app

# 5. Commit a Git
git add new-app/
git commit -m "Add new-app configuration"
```

8.2.3 Sincronizar con Git

```
cd ~/dotfiles

# Después de cambios
git add .
git commit -m "Update nvim configuration"
git push origin main

# En otra máquina
git pull origin main
stow nvim  # o stow -R nvim si ya estaba instalado
```

8.3 Manejo de Archivos Sensibles

8.3.1 Estrategia 1: .gitignore

```
# ~/dotfiles/.gitignore
# Ignorar archivos sensibles

# SSH keys
.ssh/id_*
.ssh/*.pem

# Contraseñas
.netrc
.authinfo

# Tokens
.config/gh/hosts.yml
```

8.3.2 Estrategia 2: Archivos Template

```
# Crear template sin datos sensibles
# ~/dotfiles/git/.gitconfig.local.template
[user]
    name = YOUR_NAME
    email = YOUR_EMAIL

# .gitignore
.gitconfig.local

# Script de setup
#!/bin/bash
if [ ! -f ~/dotfiles/git/.gitconfig.local ]; then
    cp ~/dotfiles/git/.gitconfig.local.template \
       ~/dotfiles/git/.gitconfig.local
    echo "Edit ~/dotfiles/git/.gitconfig.local"
fi
```

8.3.3 Estrategia 3: Encriptación

```
# Usar git-crypt o similar
cd ~/dotfiles
git-crypt init

# Especificar qué encriptar
# .gitattributes
.netrc filter=git-crypt diff=git-crypt
.ssh/id_* filter=git-crypt diff=git-crypt
```

8.4 Estructura para Múltiples Hosts

```
~/dotfiles/
   common/              # Compartido entre todos
       git/
       tmux/
   desktop/             # Solo desktop
       kde/
       i3/
   laptop/              # Solo laptop
       power-management/
   server/              # Solo servers
        ssh/
```

Script de instalación por host:

```
#!/bin/bash
# install.sh
```

```
HOSTNAME=$(hostname)

# Instalar común
cd ~/dotfiles/common
stow */

# Instalar específico del host
case "$HOSTNAME" in
    desktop-main)
        cd ~/dotfiles/desktop
        stow */
        ;;
    laptop-work)
        cd ~/dotfiles/laptop
        stow */
        ;;
    server-*)
        cd ~/dotfiles/server
        stow */
        ;;
esac
```

9 Ignore Lists

9.1 Tipos de Ignore Lists

9.1.1 Built-in (Predeterminado)

Stow ignora automáticamente:

```
RCS
.+,v
CVS
\.\#.+        # CVS conflict files / emacs lock files
\.cvsignore
\.svn
_darcs
\.hg
\.git
\.gitignore
\.gitmodules
.+~           # emacs backup files
\#.*\#        # emacs autosave files
^/README.*
^/LICENSE.*
^/COPYING
```

9.1.2 Global Ignore List

Archivo: `~/.stow-global-ignore`

```
# ~/.stow-global-ignore

# Archivos de respaldo
.*\.bak
.*\.old
.*\.orig

# Temporales
.*\.swp
.*\.tmp

# OS específicos
\.DS_Store
Thumbs\.db

# IDEs
\.idea
\.vscode

# Build artifacts
node_modules
__pycache__
*.pyc
```

9.1.3 Package-Local Ignore List

Archivo: `<package>/.stow-local-ignore`

```
# ~/dotfiles/nvim/.stow-local-ignore

# Plugin managers
^/\.config/nvim/plugin/packer_compiled\.lua

# Cache
^/\.config/nvim/.*\.cache/

# Logs
^/\.config/nvim/.*\.log

# Lazy-lock
^/\.config/nvim/lazy-lock\.json
```

9.2 Sintaxis de Ignore Lists

9.2.1 Reglas de Matching

1. Expresiones con / (path completo):

```
# Match contra path completo desde raíz del paquete
^/README.*           # README en raíz
^/\.config/nvim/cache/  # Directorio cache específico
```

2. Expresiones sin / (basename):

```
# Match contra nombre del archivo/directorio
README.*      # Cualquier README en cualquier ubicación
.*\.log       # Archivos .log en cualquier ubicación
```

9.2.2 Ejemplos Prácticos

Ejemplo 1: Ignorar documentación:

```
# .stow-local-ignore
^/README.*
^/LICENSE.*
^/CHANGELOG.*
^/docs/
```

Ejemplo 2: Ignorar archivos temporales:

```
# .stow-local-ignore
.*\.swp$
.*\.swo$
.*~$
\#.*\#$
```

Ejemplo 3: Ignorar por aplicación:

```
# nvim/.stow-local-ignore
^/\.config/nvim/plugin/
^/\.config/nvim/.*\.cache/
lazy-lock\.json

# zsh/.stow-local-ignore
\.zcompdump
\.zsh_history
```

9.3 Precedencia de Ignore Lists

```
1. .stow-local-ignore (en paquete)
     ↓ (si no existe)
2. ~/.stow-global-ignore
     ↓ (si no existe)
3. Built-in ignore list
```

9.4 Opción `--ignore` en CLI

```
# Ignorar específicos para esta ejecución
stow --ignore='.*\.orig' --ignore='.*\.dist' nvim

# Equivalente a expresión OR
stow --ignore='.*\.orig|.*\.dist' nvim

# Combina con ignore lists existentes
```

10 Opciones Avanzadas

10.1 Tree Folding Control

10.1.1 `--no-folding`

Desactiva tree folding completamente.
Sin –no-folding (default):

```
# Resultado:
~/.config -> dotfiles/nvim/.config/
```

Con –no-folding:

```
# Resultado:
~/.config/              # directorio real
    nvim -> ../dotfiles/nvim/.config/nvim/
```

Uso:

```
stow --no-folding nvim
```

10.2 Adopt Mode

10.2.1 `--adopt`

ADVERTENCIA: Modifica el contenido del stow directory.
Escenario:

```
# Tienes configuración existente:
~/.config/nvim/init.lua

# Quieres adoptarla en tu paquete:
~/dotfiles/nvim/.config/nvim/  (vacío)

# Comando:
cd ~/dotfiles
stow --adopt nvim

# Resultado:
# 1. ~/.config/nvim/init.lua → movido a ~/dotfiles/nvim/.config/nvim/init.lua
# 2. ~/.config/nvim/init.lua → se convierte en symlink
```

Uso con Git:

```
# 1. Adoptar archivos
stow --adopt nvim

# 2. Ver diferencias
cd nvim
git diff

# 3. Decidir qué mantener
git add -p  # Añadir selectivamente
# o
git checkout HEAD -- .  # Descartar cambios adoptados
```

10.3 Defer y Override

10.3.1 `--defer`

Evita stowing si el archivo ya está stowed por otro paquete.
Escenario:

```
# paquete-a tiene:
paquete-a/
    .config/
        shared/
            config.txt

# paquete-b tiene:
paquete-b/
    .config/
        shared/
            config.txt

# Instalar A primero:
stow paquete-a  # OK

# Instalar B con defer:
stow --defer='.config/shared/config.txt' paquete-b
# B no sobrescribirá config.txt de A
```

10.3.2 `--override`

Fuerza stowing incluso si ya existe symlink de otro paquete.
Escenario:

```
# Mismo escenario de arriba

# Instalar B con override:
stow --override='.config/shared/' paquete-b
# B sobrescribirá todos los archivos en .config/shared/
```

10.4 Dotfiles Mode

10.4.1 `--dotfiles`

Transforma `dot-` en `.` al hacer stow.
Uso:

```
# Estructura del paquete:
bash/
    dot-bashrc
    dot-bash_profile
    dot-config/
        bash/
            aliases.bash

# Stow con --dotfiles:
stow --dotfiles bash

# Resultado:
~/.bashrc -> dotfiles/bash/dot-bashrc
~/.bash_profile -> dotfiles/bash/dot-bash_profile
~/.config/ ...
```

Ventajas:

- Mantiene paquetes visibles (no ocultos por `.`)
- Más fácil navegar en GUI
- Mejor para Git

Desventajas:

- Necesita usar `--dotfiles` siempre
- Puede confundir
- No estándar

Recomendación: Usar nombres normales con `.` en vez de `dot-`.

10.5 Multiple Stow Directories

Puedes tener múltiples stow directories para diferentes propósitos.
Ejemplo:

```
# Estructura:
~/dotfiles/          # Personal configs
    nvim/

~/work-dotfiles/     # Work configs
    nvim/

# Marcar como stow directories:
```

```
touch ~/dotfiles/.stow
touch ~/work-dotfiles/.stow

# Stow desde diferentes directorios:
cd ~/dotfiles && stow nvim
cd ~/work-dotfiles && stow nvim
```

.stow file: Indica que un directorio es stow directory, protegiéndolo de operaciones de unstow.

11 Integración con Git

11.1 Estructura de Repositorio

```
~/dotfiles/
   .git/
   .gitignore
   .stowrc
   README.md
   LICENSE
   install.sh
   uninstall.sh
   check-stow.sh
   zsh/
       .stow-local-ignore
       .zshrc
       .zshenv
   nvim/
       .stow-local-ignore
       .config/
           nvim/
   ... (más paquetes)
```

11.2 .gitignore Completo

```
# ~/dotfiles/.gitignore

# ========================================
# BACKUPS
# ========================================
*~
*.bak
*.old
*.orig
*.swp
*.swo
```

```
# ========================================
# HISTORIA Y DATOS SENSIBLES
# ========================================

# Shell history (puede contener comandos con passwords)
**/.zsh_history
**/.bash_history
**/.history

# Credenciales
.netrc
.authinfo
**/.ssh/id_*
**/.ssh/*.pem

# Tokens
**/.config/gh/hosts.yml

# ========================================
# CACHE Y TEMPORALES
# ========================================

# Directorios de cache
**/.cache/
**/__pycache__/
**/node_modules/

# Compilados
*.pyc
*.zwc
.zcompdump*

# Logs
**/*.log

# ========================================
# ARCHIVOS DE SISTEMA
# ========================================
.DS_Store
Thumbs.db
desktop.ini

# ========================================
# STOW
# ========================================
.stow
```

```
# ======================================
# APLICACIONES ESPECÍFICAS
# ======================================

# Zotero (database muy grande)
zotero/.zotero/zotero/*/zotero.sqlite*
zotero/.zotero/zotero/*/storage/

# VSCode
vscode/.config/Code/User/workspaceStorage/
vscode/.config/Code/CachedData/
vscode/.config/Code/logs/

# Obsidian
obsidian/Documents/thoughts/.obsidian/workspace
obsidian/Documents/thoughts/.obsidian/workspace.json

# KDE
kde/.config/session/
kde/.cache/
```

11.3 Commits Best Practices

```
# Commits semánticos

# Agregar nueva aplicación
git commit -m "feat(tmux): Add tmux configuration"

# Actualizar configuración
git commit -m "chore(nvim): Update LSP settings"

# Fix
git commit -m "fix(zsh): Correct path to starship"

# Documentación
git commit -m "docs: Update README with stow instructions"

# Refactor
git commit -m "refactor(shell): Reorganize shell configs"
```

11.4 Branches Strategy

```
# Main branch
main  # Configuración estable

# Feature branches
```

```
feature/add-tmux-config
feature/new-nvim-setup

# Experimental
experiment/test-fish-shell
experiment/new-colorscheme

# Host-specific
host/desktop-main
host/laptop-work
host/server-prod
```

11.5 Tags para Versiones

```
# Tagear versiones estables
git tag -a v1.0.0 -m "Stable dotfiles v1.0.0"
git push origin v1.0.0

# Ver tags
git tag -l

# Checkout a tag
git checkout v1.0.0
```

11.6 Submodules para Plugins

```
# Agregar plugin como submodule
cd ~/dotfiles/nvim/.config/nvim
git submodule add https://github.com/user/plugin.git pack/plugins/start/plugin

# Actualizar submodules
git submodule update --init --recursive

# Pull con submodules
git pull --recurse-submodules
```

11.7 GitHub Actions para Validación

```
# .github/workflows/validate.yml

name: Validate Dotfiles

on: [push, pull_request]

jobs:
```

```
  validate:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Install stow
      run: sudo apt-get install -y stow

    - name: Test stow (dry run)
      run: |
        cd $GITHUB_WORKSPACE
        stow -nv */

    - name: Check for sensitive data
      run: |
        # Verificar que no haya claves SSH
        if find . -name "id_rsa" -o -name "id_ed25519"; then
          echo "ERROR: SSH keys found!"
          exit 1
        fi
```

12 Troubleshooting

12.1 Problema 1: Conflictos al Stow

Error:

```
WARNING! stowing nvim would cause conflicts:
  * existing target is neither a link nor a directory: .config/nvim/init.lua
All operations aborted.
```

Causa: Ya existe un archivo/directorio en el target que no es un symlink de Stow.
Soluciones:
Opción 1: Hacer backup y eliminar

```
# Backup
cp ~/.config/nvim/init.lua ~/.config/nvim/init.lua.backup

# Eliminar
rm ~/.config/nvim/init.lua

# Stow
stow nvim
```

Opción 2: Usar –adopt (cuidado)

```
stow --adopt nvim
# Mueve el archivo al paquete y crea symlink
```

Opción 3: Verificar y resolver manualmente

```
# Ver qué está causando conflicto
stow -nv nvim

# Resolver caso por caso
```

12.2 Problema 2: Symlinks Rotos

Error:

```
ls -la ~/.zshrc
# lrwxrwxrwx ... .zshrc -> dotfiles/zsh/.zshrc (broken)
```

Causa: El paquete fue movido o eliminado.
Soluciones:
Opción 1: Restow

```
cd ~/dotfiles
stow -R zsh
```

Opción 2: Desinstalar y reinstalar

```
stow -D zsh  # Limpia symlinks rotos
stow zsh     # Crea nuevos
```

Opción 3: Encontrar todos los symlinks rotos

```
# Encontrar symlinks rotos en HOME
find ~/ -xtype l

# Eliminar symlinks rotos de Stow
find ~/ -xtype l -lname '*dotfiles*' -delete
```

12.3 Problema 3: Directorio No Vacío

Error:

```
BUG in find_stowed_path? Absolute/relative mismatch
```

Causa: Stow está confundido por la estructura de directorios.
Solución:

```
# Verificar que estás en el stow directory
pwd  # Debe ser ~/dotfiles

# Verificar estructura del paquete
tree nvim

# Usar paths correctos
cd ~/dotfiles
stow -t ~ nvim
```

12.4 Problema 4: Tree Folding Inesperado

Problema: Stow crea symlink a directorio completo en lugar de entrar y enlazar archivos.

Ejemplo:

```
# Esperado:
~/.config/
    nvim/ (directorio)
        init.lua -> ~/dotfiles/nvim/.config/nvim/init.lua

# Obtenido:
~/.config/
    nvim -> ~/dotfiles/nvim/.config/nvim/ (symlink a directorio)
```

Causa: Stow hace tree folding por defecto para minimizar symlinks.
Solución si no lo quieres:

```
# Usar --no-folding
stow --no-folding nvim

# O desplegar manualmente
stow -D nvim  # Remover
mkdir -p ~/.config/nvim  # Crear directorio
stow --no-folding nvim  # Stow sin folding
```

12.5 Problema 5: Permiso Denegado

Error:

```
cannot stow: permission denied
```

Causa: No tienes permisos para crear symlinks en target directory.
Soluciones:
Para /usr/local:

```
# Cambiar ownership
sudo chown -R $USER:$USER /usr/local

# O usar sudo (no recomendado)
sudo stow -t /usr/local myapp
```

Para HOME:

```
# Verificar ownership
ls -ld ~
# drwxr-xr-x 50 user user ...

# Si no eres owner:
sudo chown -R $USER:$USER ~
```

12.6 Problema 6: Stow No Encuentra Paquete

Error:

```
stow: Cannot read package description: No such file or directory
```

Causa: No estás en el stow directory o el paquete no existe.
Solución:

```
# Verificar ubicación
pwd

# Listar paquetes disponibles
ls -d */

# Cambiar a stow directory
cd ~/dotfiles

# Stow
stow nvim
```

12.7 Problema 7: .stowrc No Se Aplica

Problema: Las opciones en .stowrc no se usan.
Causas y soluciones:
1. Archivo en ubicación incorrecta:

```
# .stowrc debe estar en:
# - Directorio actual (donde ejecutas stow)
# - O ~/

# Verificar:
ls -la .stowrc
ls -la ~/.stowrc
```

2. Sintaxis incorrecta:

```
# Correcto:
--target=/home/user

# Incorrecto:
target=/home/user  # Sin --
```

3. Variables no expandidas:

```
# Use $HOME con comillas si es necesario
--target=$HOME
```

12.8 Problema 8: Stow Muy Lento

Causa: Directorios muy grandes o muchos archivos.
Soluciones:
1. Usar ignore lists:

```
# Ignorar directorios grandes
# ~/.stow-global-ignore
node_modules
__pycache__
.cache
storage
```

2. Evitar stowing todo junto:

```
# En lugar de:
stow */  # Lento si hay muchos paquetes

# Hacer:
stow nvim zsh git  # Solo los necesarios
```

3. Simplificar estructura:

```
# Dividir paquetes grandes en paquetes más pequeños
```

13 Scripts de Automatización

13.1 Script 1: install.sh Completo

Ya proporcioné un ejemplo arriba. Aquí una versión más robusta:

```
#!/bin/bash
# ~/dotfiles/install.sh

set -e

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
DOTFILES="$SCRIPT_DIR"
BACKUP_DIR="$HOME/dotfiles-backup-$(date +%Y%m%d-%H%M%S)"
LOG_FILE="$DOTFILES/install.log"

# Colores
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
BLUE='\033[0;34m'
NC='\033[0m'

# Logging
```

```
log() {
    echo -e "$1" | tee -a "$LOG_FILE"
}

log_info() {
    log "${BLUE}[$(date +'%Y-%m-%d %H:%M:%S')]${NC} ${GREEN}[INFO]${NC} $1"
}

log_warn() {
    log "${BLUE}[$(date +'%Y-%m-%d %H:%M:%S')]${NC} ${YELLOW}[WARN]${NC} $1"
}

log_error() {
    log "${BLUE}[$(date +'%Y-%m-%d %H:%M:%S')]${NC} ${RED}[ERROR]${NC} $1"
}

# Verificar dependencias
check_dependencies() {
    log_info "Verificando dependencias..."

    if ! command -v stow &> /dev/null; then
        log_error "Stow no está instalado"
        read -p "¿Instalar stow? [Y/n] " -n 1 -r
        echo
        if [[ ! $REPLY =~ ^[Nn]$ ]]; then
            if command -v apt &> /dev/null; then
                sudo apt update && sudo apt install -y stow
            elif command -v pacman &> /dev/null; then
                sudo pacman -S stow
            elif command -v brew &> /dev/null; then
                brew install stow
            else
                log_error "No se pudo instalar stow automáticamente"
                exit 1
            fi
        else
            exit 1
        fi
    fi

    log_info "  Dependencias OK"
}

# Backup de archivo/directorio existente
backup_if_exists() {
    local path="$1"
    local name="$2"
```

```
    if [ -e "$path" ] && [ ! -L "$path" ]; then
        log_warn "Existe: $path"
        mkdir -p "$BACKUP_DIR"
        cp -r "$path" "$BACKUP_DIR/"
        log_info "Backup: $name → $BACKUP_DIR/"
        return 0
    fi
    return 1
}

# Verificar conflictos antes de stow
check_conflicts() {
    local package="$1"

    if stow -nv "$package" 2>&1 | grep -q "WARNING\|ERROR"; then
        return 1
    fi
    return 0
}

# Stow paquete con manejo de errores
stow_package() {
    local package="$1"
    local force="${2:-false}"

    if [ ! -d "$package" ]; then
        log_error "Paquete no existe: $package"
        return 1
    fi

    log_info "Procesando: $package"

    # Check conflicts
    if ! check_conflicts "$package"; then
        log_warn "Conflictos detectados en: $package"

        if [ "$force" = "true" ]; then
            log_info "Forzando instalación..."
            # Aquí podrías implementar lógica de backup automático
        else
            read -p "¿Continuar? [y/N] " -n 1 -r
            echo
            if [[ ! $REPLY =~ ^[Yy]$ ]]; then
                log_error "Saltado: $package"
                return 1
            fi
        fi
```

```
    fi

    # Stow
    if stow -v "$package"; then
        log_info "  Instalado: $package"
        return 0
    else
        log_error "  Error al instalar: $package"
        return 1
    fi
}

# Mostrar ayuda
show_help() {
    cat << EOF
Uso: $0 [opciones] [paquetes...]

Opciones:
  -h, --help          Mostrar esta ayuda
  -a, --all           Instalar todos los paquetes
  -f, --force         Forzar instalación (saltear prompts)
  -l, --list          Listar paquetes disponibles
  -d, --dry-run       Simular sin hacer cambios

Ejemplos:
  $0 nvim zsh git     # Instalar paquetes específicos
  $0 --all            # Instalar todo
  $0 --list           # Ver paquetes disponibles
EOF
}

# Listar paquetes disponibles
list_packages() {
    log_info "Paquetes disponibles:"
    cd "$DOTFILES"
    for package in */; do
        package=${package%/}
        if [ "$package" != ".git" ] && [ -d "$package" ]; then
            echo "  - $package"
        fi
    done
}

# Main
main() {
    local force=false
    local dry_run=false
```

```
    local packages=()

    # Parse argumentos
    while [[ $# -gt 0 ]]; do
        case $1 in
            -h|--help)
                show_help
                exit 0
                ;;
            -a|--all)
                cd "$DOTFILES"
                packages=($(ls -d */ | sed 's#/#' | grep -v '^\.'))
                shift
                ;;
            -f|--force)
                force=true
                shift
                ;;
            -l|--list)
                list_packages
                exit 0
                ;;
            -d|--dry-run)
                dry_run=true
                shift
                ;;
            *)
                packages+=("$1")
                shift
                ;;
        esac
    done

    # Verificar que hay paquetes para instalar
    if [ ${#packages[@]} -eq 0 ]; then
        log_error "No se especificaron paquetes"
        show_help
        exit 1
    fi

    # Iniciar log
    log_info "=== Instalación de Dotfiles ==="
    log_info "Directorio: $DOTFILES"
    log_info "Paquetes: ${packages[*]}"

    # Verificar dependencias
    check_dependencies
```

```
    # Cambiar a dotfiles directory
    cd "$DOTFILES" || exit 1

    # Dry run si se especificó
    if [ "$dry_run" = true ]; then
        log_info "=== DRY RUN ==="
        for package in "${packages[@]}"; do
            log_info "Simulating: $package"
            stow -nv "$package" || true
        done
        exit 0
    fi

    # Instalar paquetes
    local success=0
    local failed=0

    for package in "${packages[@]}"; do
        if stow_package "$package" "$force"; then
            ((success++))
        else
            ((failed++))
        fi
    done

    # Resumen
    log_info ""
    log_info "=== Resumen ==="
    log_info "Exitosos: $success"
    if [ $failed -gt 0 ]; then
        log_warn "Fallidos: $failed"
    fi

    if [ -d "$BACKUP_DIR" ]; then
        log_info "Backups en: $BACKUP_DIR"
    fi

    log_info "Log completo en: $LOG_FILE"
}

# Ejecutar
main "$@"
```

13.2 Script 2: update.sh

```
#!/bin/bash
# ~/dotfiles/update.sh

set -e

DOTFILES="$HOME/dotfiles"

cd "$DOTFILES"

echo "  Actualizando dotfiles..."

# Pull latest changes
git pull origin main

# Restow todos los paquetes instalados
for package in */; do
    package=${package%/}

    # Verificar si está stowed
    if find "$HOME" -maxdepth 2 -type l -lname "*$DOTFILES/$package/*" 2>/dev/null |
        echo "  Restowing $package..."
        stow -R "$package"
    fi
done

echo "  Actualización completa!"
```

13.3 Script 3: check.sh

```
#!/bin/bash
# ~/dotfiles/check.sh

DOTFILES="$HOME/dotfiles"

echo "  Estado de paquetes:"
echo "====================="

cd "$DOTFILES"

for package in */; do
    package=${package%/}

    if [ "$package" = ".git" ]; then
        continue
    fi

    # Buscar primer archivo del paquete
```

```
    first_file=$(find "$package" -type f -o -type l | head -1)

    if [ -z "$first_file" ]; then
        echo "   $package (vacío)"
        continue
    fi

    # Convertir a path en HOME
    home_path="$HOME/${first_file#$package/}"

    if [ -L "$home_path" ]; then
        target=$(readlink "$home_path")
        if [[ "$target" == *"$DOTFILES/$package"* ]]; then
            echo "  $package"
        else
            echo "   $package (symlink apunta a: $target)"
        fi
    elif [ -e "$home_path" ]; then
        echo "  $package (existe pero no es symlink)"
    else
        echo "  $package (no instalado)"
    fi
done

# Verificar symlinks rotos
echo ""
echo "  Verificando symlinks rotos..."
broken_links=$(find "$HOME" -maxdepth 3 -xtype l -lname "*$DOTFILES/*" 2>/dev/null)

if [ -z "$broken_links" ]; then
    echo "  No hay symlinks rotos"
else
    echo "   Symlinks rotos encontrados:"
    echo "$broken_links"
fi
```

13.4 Script 4: clean.sh

```
#!/bin/bash
# ~/dotfiles/clean.sh

DOTFILES="$HOME/dotfiles"

echo "  Limpiando symlinks huérfanos..."

# Encontrar symlinks rotos que apuntan a dotfiles
find "$HOME" -maxdepth 3 -xtype l -lname "*$DOTFILES/*" 2>/dev/null | while read -r b
```

```
    echo "Eliminando: $broken_link"
    rm "$broken_link"
done

echo "  Limpieza completa!"
```

14 Casos de Uso Prácticos

14.1 Caso 1: Crear Dotfiles desde Cero (Primera Vez)

Escenario: Nunca has usado Stow, quieres empezar desde cero organizando tus configuraciones.

Objetivo: Crear estructura de dotfiles, migrar configs existentes, versionar con Git.

14.1.1 Paso 1: Preparación

```
# 1.1 Instalar herramientas necesarias
sudo pacman -S stow git zsh starship  # Arch/Archcraft
# o
sudo apt install stow git zsh        # Kubuntu/Debian

# 1.2 Verificar instalación
stow --version
git --version

# 1.3 Crear directorio para dotfiles
mkdir -p ~/dotfiles
cd ~/dotfiles

# 1.4 Inicializar Git
git init
git branch -M main

# 1.5 Configurar Git (si no está configurado)
git config user.name "Edison Achalma"
git config user.email "achalmaedison@gmail.com"
```

14.1.2 Paso 2: Crear Estructura de Paquetes

```
# Crear paquetes para cada aplicación
cd ~/dotfiles

# Git
mkdir -p git
# Shell (Zsh)
mkdir -p shell
# Terminal (Konsole)
```

```
mkdir -p terminal
# Editor (VSCode)
mkdir -p vscode
# KDE
mkdir -p kde
```

14.1.3 Paso 3: Migrar Configuraciones Existentes

3.1 Git (.gitconfig):

```
# Verificar que existe
ls -la ~/.gitconfig

# Mover al paquete
mv ~/.gitconfig git/

# Verificar
ls -la git/.gitconfig
```

3.2 Shell (Zsh):

```
# Crear estructura
mkdir -p shell

# Mover archivos
mv ~/.zshrc shell/
mv ~/.zshenv shell/ 2>/dev/null || true  # Si existe

# Si tienes starship
mv ~/.config/starship.toml shell/ 2>/dev/null || true

# Verificar
tree shell/
# shell/
#    .zshrc
#    .zshenv
```

3.3 Terminal (Konsole):

```
# Crear estructura que replica HOME
mkdir -p terminal/.config

# Mover config de Konsole
mv ~/.config/konsolerc terminal/.config/

# Si tienes perfiles personalizados
cp -r ~/.local/share/konsole terminal/.local/share/ 2>/dev/null || true
```

```
# Verificar
tree terminal/
# terminal/
#    .config/
#        konsolerc
```

3.4 VSCode:

```
# Crear estructura
mkdir -p vscode/.config/Code/User

# Mover settings
mv ~/.config/Code/User/settings.json vscode/.config/Code/User/
mv ~/.config/Code/User/keybindings.json vscode/.config/Code/User/

# Snippets
mv ~/.config/Code/User/snippets vscode/.config/Code/User/ 2>/dev/null || true

# Verificar
tree vscode/.config/Code/User/
```

3.5 KDE Plasma:

```
# Crear estructura
mkdir -p kde/.config

# Mover configuraciones principales
mv ~/.config/kdeglobals kde/.config/
mv ~/.config/dolphinrc kde/.config/
mv ~/.config/kwinrc kde/.config/
mv ~/.config/plasmarc kde/.config/
mv ~/.config/plasma-org.kde.plasma.desktop-appletsrc kde/.config/
mv ~/.config/mimeapps.list kde/.config/

# Verificar
ls kde/.config/
```

14.1.4 Paso 4: Crear Ignore Lists

4.1 Global ignore:

```
cat > ~/.stow-global-ignore << 'EOF'
# Backups
.*~
.*\.bak
.*\.old
.*\.orig
.*\.swp
```

```
# Historia
\.zsh_history
\.bash_history

# Cache
\.cache
__pycache__

# Sistema
\.DS_Store
Thumbs\.db

# Git
\.git
\.gitignore

# Documentación
^/README.*
^/LICENSE.*
EOF
```

4.2 Ignore por paquete (shell):

```
cat > shell/.stow-local-ignore << 'EOF'
# Historia (datos sensibles)
^/\.zsh_history
^/\.bash_history

# Cache compilado
\.zcompdump
EOF
```

4.3 Ignore para VSCode:

```
cat > vscode/.stow-local-ignore << 'EOF'
# Cache y logs
^/\.config/Code/CachedData/
^/\.config/Code/logs/
^/\.config/Code/User/workspaceStorage/

# Backups automáticos
^/\.config/Code/Backups/
EOF
```

14.1.5 Paso 5: Crear .gitignore

```
cat > .gitignore << 'EOF'
# ========================================
# BACKUPS
# ========================================
*~
*.bak
*.old
*.orig
*.swp
*.swo

# ========================================
# DATOS SENSIBLES
# ========================================
# Historia de shells
**/.zsh_history
**/.bash_history

# SSH keys
**/.ssh/id_*
**/.ssh/*.pem

# Credenciales
.netrc
.authinfo

# ========================================
# CACHE Y TEMPORALES
# ========================================
**/.cache/
**/__pycache__/
*.pyc
.zcompdump*

# ========================================
# STOW
# ========================================
.stow

# ========================================
# LOGS
# ========================================
**/*.log
*.log

# ========================================
# SISTEMA
```

```
# ======================================
.DS_Store
Thumbs.db
desktop.ini
EOF
```

14.1.6 Paso 6: Crear .stowrc

```
cat > .stowrc << 'EOF'
# Target es siempre HOME
--target=$HOME

# Ignorar archivos comunes
--ignore='^\.git'
--ignore='^README.*'
--ignore='^LICENSE.*'
--ignore='\.gitignore$'
--ignore='.*\.swp$'
--ignore='.*~$'
--ignore='^install\.sh$'
--ignore='^\.stowrc$'
EOF
```

14.1.7 Paso 7: Instalar con Stow (Primera Vez)

```
# Navegar a dotfiles
cd ~/dotfiles

# Dry run primero para cada paquete
stow -nv git
stow -nv shell
stow -nv terminal
stow -nv vscode
stow -nv kde

# Si todo OK, instalar realmente
stow git shell terminal vscode kde

# Verificar symlinks
ls -la ~/.gitconfig
# lrwxrwxrwx ... .gitconfig -> dotfiles/git/.gitconfig

ls -la ~/.zshrc
# lrwxrwxrwx ... .zshrc -> dotfiles/shell/.zshrc

ls -la ~/.config/konsolerc
# lrwxrwxrwx ... konsolerc -> ../dotfiles/terminal/.config/konsolerc
```

14.1.8 Paso 8: Verificar que Todo Funciona

```
# 8.1 Verificar Git
git config --list | head -5

# 8.2 Verificar Zsh
cat ~/.zshrc | head -10

# 8.3 Verificar VSCode
cat ~/.config/Code/User/settings.json | head -10

# 8.4 Abrir aplicaciones para probar
code  # VSCode debe cargar tu config
konsole  # Konsole debe tener tu configuración
```

14.1.9 Paso 9: Crear Scripts de Ayuda

9.1 Script de instalación:

```
cat > install.sh << 'EOF'
#!/bin/bash
set -e

DOTFILES="$HOME/dotfiles"

echo "  Instalando dotfiles..."

cd "$DOTFILES"

# Lista de paquetes
PACKAGES=(
    "git"
    "shell"
    "terminal"
    "vscode"
    "kde"
)

# Instalar cada paquete
for pkg in "${PACKAGES[@]}"; do
    echo "  Instalando $pkg..."
    stow "$pkg"
done

echo "  ¡Instalación completa!"
EOF

chmod +x install.sh
```

9.2 Script de verificación:

```
cat > check.sh << 'EOF'
#!/bin/bash

DOTFILES="$HOME/dotfiles"

echo "  Verificando dotfiles..."
echo ""

cd "$DOTFILES"

for pkg in */; do
    pkg=${pkg%/}

    # Buscar primer archivo
    first_file=$(find "$pkg" -type f | head -1)

    if [ -z "$first_file" ]; then
        continue
    fi

    # Path en HOME
    home_path="$HOME/${first_file#$pkg/}"

    if [ -L "$home_path" ]; then
        echo "  $pkg"
    else
        echo "  $pkg (no instalado)"
    fi
done
EOF

chmod +x check.sh
```

14.1.10 Paso 10: Crear Repositorio en GitHub

```
# 10.1 Agregar todo a Git
cd ~/dotfiles
git add .

# 10.2 Commit inicial
git commit -m "Initial commit: Estructura básica de dotfiles

- Git configuration
- Zsh/Starship setup
- Konsole terminal config
- VSCode settings
```

```
- KDE Plasma configuration"

# 10.3 Crear repo en GitHub (vía navegador o gh CLI)
# Opción A: Navegador
# Ve a https://github.com/new
# Nombre: .dotfiles
# Descripción: "Dotfiles para Archcraft/Kubuntu con Stow"
# Público o Privado
# NO inicializar con README (ya lo tienes)

# Opción B: GitHub CLI
gh repo create .dotfiles --public --source=. --remote=origin

# 10.4 Agregar remote y push
git remote add origin https://github.com/achalmaedison/.dotfiles.git
git push -u origin main
```

14.1.11 Paso 11: Crear README.md

```
cat > README.md << 'EOF'

# Dotfiles

Configuraciones personales para Archcraft/Kubuntu gestionadas con GNU Stow.

## Estructura

``
~/dotfiles/
   git/          # Git config
   shell/        # Zsh + Starship
   terminal/     # Konsole
   vscode/       # Visual Studio Code
   kde/          # KDE Plasma
``

## Instalación

``bash
# Clonar
git clone https://github.com/achalmaedison/.dotfiles.git ~/dotfiles

# Instalar Stow
sudo pacman -S stow  # Arch
# o
sudo apt install stow  # Debian/Ubuntu
```

```
# Instalar todo
cd ~/dotfiles
./install.sh

# O instalar selectivo
stow git shell terminal
``

## Actualizar

``bash
cd ~/dotfiles
git pull
stow -R */
``

## Requisitos

- stow
- git
- zsh
- starship (opcional)
- VSCode (opcional)
- KDE Plasma (opcional)
EOF

git add README.md
git commit -m "docs: Add README"
git push
```

14.2 Caso 2: Replicar Dotfiles en Laptop Nueva

Escenario: Acabas de comprar/instalar una laptop nueva con Archcraft y quieres replicar tu setup completo.

Objetivo: Clonar tu repo de dotfiles e instalar todo en la nueva máquina.

14.2.1 Paso 1: Preparar Nueva Máquina

```
# 1.1 Actualizar sistema (Archcraft/Arch)
sudo pacman -Syu

# 1.2 Instalar herramientas base
sudo pacman -S git stow zsh base-devel

# 1.3 Verificar HOME vacío (opcional)
ls -la ~/ | grep "^\." | wc -l
# Deberías ver solo archivos básicos del sistema
```

14.2.2 Paso 2: Backup de Configs Existentes (Precaución)

```
# 2.1 Crear directorio de backup
mkdir -p ~/dotfiles-backup-$(date +%Y%m%d)
BACKUP_DIR=~/dotfiles-backup-$(date +%Y%m%d)

# 2.2 Backup de archivos que podrían existir
cp ~/.zshrc "$BACKUP_DIR/" 2>/dev/null || true
cp ~/.gitconfig "$BACKUP_DIR/" 2>/dev/null || true
cp -r ~/.config/Code "$BACKUP_DIR/" 2>/dev/null || true

echo "Backup guardado en: $BACKUP_DIR"
ls -la "$BACKUP_DIR"
```

14.2.3 Paso 3: Clonar Repositorio

```
# 3.1 Clonar tu repo
cd ~
git clone https://github.com/achalmaedison/.dotfiles.git dotfiles

# 3.2 Verificar contenido
cd dotfiles
ls -la

# Deberías ver:
# git/
# shell/
# terminal/
# vscode/
# kde/
# install.sh
# .gitignore
# README.md
```

14.2.4 Paso 4: Revisar y Ajustar (Si Necesario)

```
# 4.1 Ver qué paquetes hay
ls -d */
# git/  kde/  shell/  terminal/  vscode/

# 4.2 Ver estructura de un paquete
tree shell/
# shell/
#    .zshrc
#    .zshenv

# 4.3 (Opcional) Editar configs antes de instalar
```

```
# Por ejemplo, cambiar username en git
nano git/.gitconfig
```

14.2.5 Paso 5: Instalar Dependencias

```
# 5.1 Aplicaciones de tu setup
sudo pacman -S \
    zsh \
    starship \
    konsole \
    code \  # VSCode (si está en repos)
    plasma-desktop \
    dolphin \
    kate \
    okular

# 5.2 Si usas AUR (yay/paru)
# VSCode desde AUR
yay -S visual-studio-code-bin

# Starship (si no está en repos oficiales)
yay -S starship-bin

# 5.3 Verificar instalaciones
which zsh
which starship
which code
```

14.2.6 Paso 6: Dry Run (Simulación)

```
# Navegar a dotfiles
cd ~/dotfiles

# Simular instalación para ver qué pasaría
stow -nv git
stow -nv shell
stow -nv terminal
stow -nv vscode
stow -nv kde

# Verificar que no hay errores
# Si hay conflictos, verás warnings
```

14.2.7 Paso 7: Resolver Conflictos (Si Existen)

Si ves algo como:

```
WARNING! stowing shell would cause conflicts:
  * existing target is neither a link nor a directory: .zshrc
```

Resolver:

```
# Opción A: Eliminar archivo existente
rm ~/.zshrc

# Opción B: Mover a backup (más seguro)
mv ~/.zshrc ~/dotfiles-backup-$(date +%Y%m%d)/

# Luego intentar stow nuevamente
stow -nv shell
```

14.2.8 Paso 8: Instalar Todo

Opción A: Con script (recomendado):

```
cd ~/dotfiles
./install.sh
```

Opción B: Manual:

```
cd ~/dotfiles

# Instalar uno por uno
stow git
echo "  Git instalado"

stow shell
echo "  Shell instalado"

stow terminal
echo "  Terminal instalado"

stow vscode
echo "  VSCode instalado"

stow kde
echo "  KDE instalado"
```

Opción C: Todo de una vez:

```
cd ~/dotfiles
stow git shell terminal vscode kde
```

14.2.9 Paso 9: Verificar Instalación

```
# 9.1 Verificar symlinks creados
ls -la ~/.gitconfig
# lrwxrwxrwx ... .gitconfig -> dotfiles/git/.gitconfig

ls -la ~/.zshrc
# lrwxrwxrwx ... .zshrc -> dotfiles/shell/.zshrc

ls -la ~/.config/Code/User/settings.json
# lrwxrwxrwx ... settings.json -> ../../../../dotfiles/vscode/.config/Code/User/setti

# 9.2 Usar script de verificación
cd ~/dotfiles
./check.sh
```

14.2.10 Paso 10: Configurar Shell

```
# 10.1 Cambiar shell a Zsh (si no lo es)
chsh -s /bin/zsh

# 10.2 Logout y login para aplicar
# O simplemente:
exec zsh

# 10.3 Verificar que Zsh cargó tu config
echo $SHELL
# /bin/zsh

# Ver prompt (si usas starship)
starship --version
```

14.2.11 Paso 11: Instalar Dependencias Específicas

11.1 Extensiones de VSCode:

```
# Si guardaste lista de extensiones
# (Opción: guardar en dotfiles)
code --list-extensions > ~/dotfiles/vscode/extensions.txt

# En nueva máquina:
while read -r ext; do
    code --install-extension "$ext"
done < ~/dotfiles/vscode/extensions.txt
```

11.2 Plugins de Zsh (si usas):

```
# Oh-My-Zsh
sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
```

```
# Zsh plugins (ejemplo: zsh-autosuggestions)
git clone https://github.com/zsh-users/zsh-autosuggestions \
    ~/.oh-my-zsh/custom/plugins/zsh-autosuggestions
```

11.3 Temas de KDE (si los tienes):

```
# Si tienes temas personalizados en dotfiles
# Instalarlos desde System Settings
```

14.2.12 Paso 12: Probar Todo

```
# 12.1 Git
git config --list | grep user
# user.name=Edison Achalma
# user.email=achalmaedison@gmail.com

# 12.2 Zsh
cat ~/.zshrc | head -5

# 12.3 VSCode
code
# Debería cargar con tu configuración

# 12.4 Konsole
konsole
# Debería usar tu configuración

# 12.5 KDE
# Logout/login para ver cambios en KDE
```

14.2.13 Paso 13: Ajustes Finales

```
# Si algo no funciona, hacer debug:

# Ver qué apunta cada symlink
find ~ -maxdepth 2 -type l -ls | grep dotfiles

# Si un symlink está roto:
stow -D paquete-con-problema
stow paquete-con-problema

# Verificar logs
journalctl --user -xe | grep -i error
```

14.3 Caso 3: Actualizar Configs y Sincronizar

Escenario: Has estado usando tus dotfiles y has hecho cambios en tu máquina principal. Quieres sincronizar con GitHub y otras máquinas.

14.3.1 Paso 1: Identificar Cambios

```
# 1.1 Ver qué archivos cambiaron
cd ~/dotfiles
git status

# Ejemplo de output:
# modified:   shell/.zshrc
# modified:   vscode/.config/Code/User/settings.json

# 1.2 Ver diferencias específicas
git diff shell/.zshrc
git diff vscode/.config/Code/User/settings.json

# 1.3 Ver todos los cambios
git diff
```

14.3.2 Paso 2: Probar Cambios Localmente

```
# Si editaste configs directamente en HOME (a través de symlinks),
# los cambios ya están en ~/dotfiles/

# 2.1 Verificar que todo funciona
source ~/.zshrc  # Para shell
code  # Abrir VSCode para verificar settings

# 2.2 Si hay problemas, revertir temporalmente
cd ~/dotfiles
git checkout -- shell/.zshrc  # Revertir cambios
# Probar de nuevo
```

14.3.3 Paso 3: Commit Cambios

```
cd ~/dotfiles

# 3.1 Agregar archivos modificados
git add shell/.zshrc
git add vscode/.config/Code/User/settings.json

# O agregar todo:
git add -A

# 3.2 Ver qué se va a commitear
git status

# 3.3 Commit con mensaje descriptivo
```

```
git commit -m "chore(shell): Update Zsh aliases and PATH

- Add alias for git status
- Update PATH to include ~/.local/bin
- Remove deprecated exports"

git commit -m "feat(vscode): Enable format on save

- Set editor.formatOnSave to true
- Add Python formatting rules
- Update keybindings for terminal"
```

14.3.4 Paso 4: Push a GitHub

```
# 4.1 Push cambios
git push origin main

# 4.2 Verificar en GitHub
# Ir a https://github.com/achalmaedison/.dotfiles
# Deberías ver tus commits recientes
```

14.3.5 Paso 5: Actualizar Otras Máquinas

En laptop/otra máquina:

```
# 5.1 Pull cambios
cd ~/dotfiles
git pull origin main

# 5.2 Los symlinks reflejan cambios automáticamente!
cat ~/.zshrc  # Ya tiene los cambios

# 5.3 Recargar configs
source ~/.zshrc  # Shell
# VSCode se recarga automáticamente

# 5.4 Si hay cambios en estructura (archivos nuevos/eliminados):
stow -R shell  # Restow para actualizar symlinks
stow -R vscode
```

14.3.6 Paso 6: Manejar Conflictos (Si Existen)

Si modificaste el mismo archivo en dos máquinas:

```
cd ~/dotfiles
git pull origin main

# Si hay conflicto:
```

```
# CONFLICT (content): Merge conflict in shell/.zshrc

# 6.1 Ver conflicto
git status
# both modified:   shell/.zshrc

# 6.2 Editar archivo
nano shell/.zshrc

# Verás marcadores:
# <<<<<<< HEAD
# (tu cambio local)
# =======
# (cambio de GitHub)
# >>>>>>> origin/main

# 6.3 Resolver manualmente, eliminar marcadores

# 6.4 Marcar como resuelto
git add shell/.zshrc
git commit -m "merge: Resolve conflict in .zshrc"
git push
```

14.4 Caso 4: Agregar Nueva Aplicación (Kate Editor)

Escenario: Instalaste Kate y quieres agregar su configuración a tus dotfiles.

14.4.1 Paso 1: Usar Kate y Configurar

```
# 1.1 Instalar Kate
sudo pacman -S kate

# 1.2 Abrir y configurar
kate

# Configurar:
# - Settings → Configure Kate
# - Cambiar tema, shortcuts, plugins, etc.
# - Cerrar Kate (configs se guardan automáticamente)
```

14.4.2 Paso 2: Localizar Archivos de Config

```
# 2.1 Archivos de configuración están en ~/.config/
ls -la ~/.config/ | grep kate
# drwxr-xr-x  - achalmaedison kate/

# 2.2 Ver qué hay dentro
```

```
ls -la ~/.config/kate/
# katerc
# externaltools/
# formatting/
# lspclient/

# 2.3 También puede haber datos en ~/.local/share/
ls -la ~/.local/share/ | grep kate
```

14.4.3 Paso 3: Crear Paquete Kate

```
# 3.1 Crear estructura que replica HOME
cd ~/dotfiles
mkdir -p kate/.config

# 3.2 Copiar configs (NO mover todavía)
cp -r ~/.config/kate kate/.config/

# 3.3 También copiar datos locales si existen
mkdir -p kate/.local/share
cp -r ~/.local/share/kate kate/.local/share/ 2>/dev/null || true

# 3.4 Verificar estructura
tree kate/
# kate/
#    .config/
#       kate/
#           katerc
#           externaltools/
#           formatting/
#           lspclient/
#    .local/
#        share/
#            kate/
```

14.4.4 Paso 4: Crear Ignore List para Kate

```
# Crear ignore para archivos que no queremos versionar
cat > kate/.stow-local-ignore << 'EOF'
# Sesiones y cache
^/\.config/kate/sessions/
^/\.local/share/kate/.*\.cache

# Logs
^/\.config/kate/.*\.log

# Archivos temporales
```

```
^/\.config/kate/.*\.tmp
^/\.config/kate/.*\.swp
EOF
```

14.4.5 Paso 5: Test Stow (Dry Run)

```
cd ~/dotfiles

# 5.1 Ver qué haría stow
stow -nv kate

# Deberías ver algo como:
# LINK: .config/kate => dotfiles/kate/.config/kate
```

14.4.6 Paso 6: Hacer Backup y Eliminar Original

```
# 6.1 Backup por seguridad
cp -r ~/.config/kate ~/backup-kate-$(date +%Y%m%d)

# 6.2 Eliminar original
rm -rf ~/.config/kate
rm -rf ~/.local/share/kate  # Si copiaste esto también

# 6.3 Verificar que se eliminó
ls ~/.config/ | grep kate
# (no debería aparecer nada)
```

14.4.7 Paso 7: Stow Kate

```
cd ~/dotfiles

# 7.1 Instalar con stow
stow kate

# 7.2 Verificar symlinks
ls -la ~/.config/ | grep kate
# lrwxrwxrwx  - achalmaedison kate -> ../dotfiles/kate/.config/kate

# 7.3 Verificar que Kate funciona
kate
# Debería abrir con tu configuración
```

14.4.8 Paso 8: Versionar con Git

```
cd ~/dotfiles

# 8.1 Agregar al staging
git add kate/

# 8.2 Commit
git commit -m "feat(kate): Add Kate editor configuration

- Custom keybindings
- LSP configuration
- Theme and appearance settings
- External tools setup"

# 8.3 Push
git push origin main
```

14.4.9 Paso 9: Actualizar README

```
cd ~/dotfiles

# Agregar Kate a la lista de paquetes
nano README.md

# Agregar:
# - kate/          # Kate editor

# Commit cambio
git add README.md
git commit -m "docs: Add Kate to README"
git push
```

14.4.10 Paso 10: Actualizar Script de Instalación

```
# Si tienes install.sh, agregar kate
nano install.sh

# Agregar "kate" a la lista de PACKAGES:
# PACKAGES=(
#     "git"
#     "shell"
#     "terminal"
#     "vscode"
#     "kde"
#     "kate"        # <-- Agregar esto
# )
```

```
git add install.sh
git commit -m "chore(scripts): Add kate to install script"
git push
```

14.5 Caso 5: Migrar de Kubuntu a Archcraft

Escenario: Usabas Kubuntu, ahora instalaste Archcraft. Quieres migrar tus dotfiles pero adaptándolos.

14.5.1 Paso 1: Evaluar Diferencias

```
# 1.1 En tu Kubuntu original, ver qué tienes
cd ~/dotfiles
ls -d */

# Ejemplo:
# git/ shell/ terminal/ vscode/ kde/ digikam/ okular/ ...

# 1.2 Identificar qué es compatible con Archcraft
#  Compatible: git, shell, vscode
#   Adaptar: kde (Archcraft puede usar i3/bspwm)
#  No necesario: apps específicas de Kubuntu
```

14.5.2 Paso 2: En Archcraft Nueva

```
# 2.1 Instalar Stow
sudo pacman -S stow git

# 2.2 Clonar dotfiles
cd ~
git clone https://github.com/achalmaedison/.dotfiles.git dotfiles
```

14.5.3 Paso 3: Crear Branch para Archcraft

```
cd ~/dotfiles

# 3.1 Crear branch específica
git checkout -b archcraft-setup

# 3.2 Ver qué paquetes hay
ls -d */
```

14.5.4 Paso 4: Instalar Paquetes Universales

```
cd ~/dotfiles

# 4.1 Paquetes que funcionan en cualquier distro
stow git
stow shell
stow terminal  # Si Archcraft usa Konsole, sino adaptar
```

14.5.5 Paso 5: Adaptar o Crear Nuevos Paquetes

5.1 Window Manager (Si Archcraft usa i3/bspwm):

```
# Crear nuevo paquete para i3 (ejemplo)
mkdir -p i3/.config/i3

# Configurar i3
i3-config-wizard
# O copiar config existente

# Mover config al paquete
mv ~/.config/i3/config i3/.config/i3/

# Stow
stow i3
```

5.2 Terminal (Si Archcraft usa otro terminal):

```
# Supongamos que Archcraft usa Alacritty en vez de Konsole

# Crear paquete
mkdir -p alacritty/.config/alacritty

# Config de Alacritty
cat > alacritty/.config/alacritty/alacritty.yml << 'EOF'
# Alacritty configuration
font:
  size: 11.0
  normal:
    family: JetBrains Mono

window:
  opacity: 0.95

colors:
  # Tu esquema de colores...
EOF

# Stow
stow alacritty
```

5.3 Polybar (Si Archcraft lo usa):

```
mkdir -p polybar/.config/polybar

# Copiar config de Archcraft default
cp /etc/polybar/config polybar/.config/polybar/

# Personalizar
nano polybar/.config/polybar/config

# Stow
stow polybar
```

14.5.6 Paso 6: No Instalar Paquetes Incompatibles

```
# NO hacer stow de paquetes específicos de Kubuntu/KDE:
# - kde/
# - plasma-org.kde.plasma.desktop-appletsrc
# - etc.

# Estos causarían errores en Archcraft
```

14.5.7 Paso 7: Commit Cambios

```
cd ~/dotfiles

# Agregar nuevos paquetes
git add i3/ alacritty/ polybar/

# Commit en branch archcraft
git commit -m "feat(archcraft): Add i3, Alacritty, Polybar configs

- i3 window manager configuration
- Alacritty terminal setup
- Polybar panel configuration"

# Push branch
git push origin archcraft-setup
```

14.5.8 Paso 8: Estrategia de Branches

Opción A: Mantener branches separadas:

```
# Branch main: Para Kubuntu
# Branch archcraft-setup: Para Archcraft

# Puedes hacer cherry-pick de commits específicos:
```

```
git checkout main
git cherry-pick <commit-hash>  # Traer cambio específico de otra branch
```

Opción B: Usar estructura de directorios:

```
# Reorganizar dotfiles:
~/dotfiles/
   common/          # Configs universales
       git/
       shell/
       vscode/
   kubuntu/         # Específicos de Kubuntu
       kde/
       konsole/
   archcraft/       # Específicos de Archcraft
       i3/
       alacritty/
       polybar/

# Instalar según distro:
cd ~/dotfiles/common && stow */
cd ~/dotfiles/archcraft && stow */
```

14.5.9 Paso 9: Script de Instalación por Distro

```
cat > install-arch.sh << 'EOF'
#!/bin/bash
# Install script para Archcraft

DOTFILES="$HOME/dotfiles"

echo "  Instalando dotfiles para Archcraft..."

# Common packages
cd "$DOTFILES/common"
stow git shell vscode

# Archcraft-specific
cd "$DOTFILES/archcraft"
stow i3 alacritty polybar rofi

echo "  ¡Instalación completa!"
EOF

chmod +x install-arch.sh
```

14.5.10 Paso 10: Mantener Ambos Sistemas

```
# Cuando hagas cambios en configs comunes (git, shell, vscode):

# 1. Hacer cambio en cualquier máquina
cd ~/dotfiles
nano common/shell/.zshrc

# 2. Commit
git add common/shell/.zshrc
git commit -m "chore(shell): Update aliases"

# 3. Push
git push origin main

# 4. En otra máquina (Kubuntu o Archcraft):
git pull origin main
# Los symlinks se actualizan automáticamente
```

14.6 Caso 6: Probar Nueva Configuración Sin Romper

Escenario: Quieres probar una nueva configuración de Neovim sin afectar tu setup actual.

14.6.1 Paso 1: Crear Branch Experimental

```
cd ~/dotfiles

# 1.1 Crear branch
git checkout -b experiment/nvim-lazyvim

# 1.2 Verificar que estás en la branch
git branch
# * experiment/nvim-lazyvim
#   main
```

14.6.2 Paso 2: Crear Paquete Alternativo

```
# 2.1 Crear nuevo paquete con nombre distinto
mkdir -p nvim-lazy/.config

# 2.2 Instalar LazyVim (ejemplo)
git clone https://github.com/LazyVim/starter nvim-lazy/.config/nvim

# 2.3 Estructura
tree nvim-lazy/.config/nvim/ -L 1
```

14.6.3 Paso 3: Desinstalar Neovim Actual

```
# 3.1 Unstow config actual (si existe)
cd ~/dotfiles
stow -D nvim 2>/dev/null || true

# 3.2 Verificar que se eliminó symlink
ls -la ~/.config/ | grep nvim
# No debería aparecer nada
```

14.6.4 Paso 4: Instalar Nueva Config

```
# 4.1 Stow nueva config
stow nvim-lazy

# 4.2 Verificar symlink
ls -la ~/.config/nvim
# lrwxrwxrwx  - achalmaedison nvim -> ../../dotfiles/nvim-lazy/.config/nvim
```

14.6.5 Paso 5: Probar

```
# 5.1 Abrir Neovim
nvim

# LazyVim se instalará automáticamente
# Probar todas las features

# 5.2 Usar por varios días
# Evaluar si te gusta
```

14.6.6 Paso 6: Decidir Qué Hacer

Opción A: Mantener nueva config (si te gustó):

```
cd ~/dotfiles

# 1. Eliminar config vieja
rm -rf nvim/  # O hacer backup

# 2. Renombrar nueva
mv nvim-lazy nvim

# 3. Restow
stow -R nvim

# 4. Commit
git add .
```

```
git commit -m "refactor(nvim): Switch to LazyVim configuration"

# 5. Merge a main
git checkout main
git merge experiment/nvim-lazyvim

# 6. Push
git push origin main

# 7. Eliminar branch experimental
git branch -d experiment/nvim-lazyvim
```

Opción B: Volver a config anterior (si no te gustó):

```
cd ~/dotfiles

# 1. Checkout a main
git checkout main

# 2. Unstow experimental
stow -D nvim-lazy

# 3. Restow original
stow nvim

# 4. Eliminar paquete experimental
rm -rf nvim-lazy/

# 5. Eliminar branch
git branch -D experiment/nvim-lazyvim
```

Opción C: Mantener ambas (para casos específicos):

```
# Tener dos configs de Neovim:
~/dotfiles/
   nvim/          # Config principal
   nvim-lazy/     # Config alternativa

# Alias en shell para cambiar:
alias nvim-main='stow -D nvim-lazy && stow nvim && nvim'
alias nvim-lazy='stow -D nvim && stow nvim-lazy && nvim'
```

14.7 Caso 7: Sincronizar Múltiples Máquinas en Tiempo Real

Escenario: Trabajas en 3 máquinas (desktop, laptop, servidor) y quieres mantener dotfiles sincronizados.

14.7.1 Configuración Inicial (Una Vez)

```
# En cada máquina:

# 1. Clonar dotfiles
cd ~
git clone https://github.com/achalmaedison/.dotfiles.git dotfiles

# 2. Instalar
cd dotfiles
./install.sh

# 3. Configurar Git con pull automático (opcional)
git config pull.rebase true  # Rebase en lugar de merge
```

14.7.2 Workflow Diario

Máquina A (Desktop) - Hacer Cambios:

```
# 1. Editar configs normalmente
nano ~/.zshrc  # Edita a través del symlink

# 2. Commit y push
cd ~/dotfiles
git add shell/.zshrc
git commit -m "chore(shell): Add new alias for docker"
git push origin main
```

Máquina B (Laptop) - Recibir Cambios:

```
# 1. Pull cambios
cd ~/dotfiles
git pull origin main

# 2. Los symlinks se actualizan automáticamente!
cat ~/.zshrc  # Ya tiene el cambio

# 3. Recargar shell
source ~/.zshrc
# O
exec zsh
```

Máquina C (Servidor) - Recibir Cambios:

```
# Mismo proceso
cd ~/dotfiles
git pull origin main
source ~/.zshrc
```

14.7.3 Automatizar con Cron (Opcional)

```
# Crear script de sync
cat > ~/dotfiles/sync.sh << 'EOF'
#!/bin/bash
cd "$HOME/dotfiles"

# Pull cambios silenciosamente
git pull origin main --quiet

# Log
echo "$(date): Dotfiles sincronizados" >> ~/dotfiles/sync.log
EOF

chmod +x ~/dotfiles/sync.sh

# Agregar a crontab (sync cada hora)
crontab -e

# Agregar línea:
0 * * * * $HOME/dotfiles/sync.sh
```

14.7.4 Manejar Conflictos Automáticamente

```
# Script más robusto
cat > ~/dotfiles/sync.sh << 'EOF'
#!/bin/bash
cd "$HOME/dotfiles"

# Stash cambios locales si existen
git stash

# Pull
git pull origin main --quiet

# Reapply stash
git stash pop

# Si hay conflictos, notificar
if [ $? -ne 0 ]; then
    notify-send "Dotfiles" "Conflicto detectado, revisar manualmente"
fi
EOF
```

14.7.5 Usar Git Hooks (Avanzado)

```
# Pre-commit hook para validar antes de commit
cat > ~/dotfiles/.git/hooks/pre-commit << 'EOF'
#!/bin/bash

# Verificar que no hay datos sensibles
if git diff --cached | grep -i "password\|secret\|token"; then
    echo "ERROR: Posible dato sensible detectado!"
    exit 1
fi

exit 0
EOF

chmod +x ~/dotfiles/.git/hooks/pre-commit
```

14.8 Caso 8: Compartir Dotfiles con Equipo/Lab

Escenario: Trabajas en un lab con múltiples usuarios y quieren compartir configuraciones base.

14.8.1 Paso 1: Crear Repo de Equipo

```
# En GitHub, crear repo:
# Nombre: lab-dotfiles
# Acceso: Privado/Público según necesidad

# Clonar
git clone https://github.com/lab/.lab-dotfiles.git ~/lab-dotfiles
```

14.8.2 Paso 2: Estructura Multi-Usuario

```
cd ~/lab-dotfiles

# Crear estructura
mkdir -p {common,users}

# Common: Configs compartidas
mkdir -p common/{git,shell,terminal}

# Users: Configs personales
mkdir -p users/{alice,bob,carlos}
```

14.8.3 Paso 3: Setup Común

```
# Git config compartido (sin user.name/email)
cat > common/git/.gitconfig << 'EOF'
[core]
    editor = nano
    autocrlf = input

[alias]
    st = status
    co = checkout
    br = branch

[push]
    default = simple
EOF

# Shell común
cat > common/shell/.zshrc << 'EOF'
# Shared Zsh configuration for Lab

# Common aliases
alias ll='ls -lah'
alias ..='cd ..'

# Lab-specific paths
export LAB_DATA="/data/lab"
export LAB_TOOLS="/opt/lab-tools"

# Source user-specific config if exists
[ -f ~/.zshrc.local ] && source ~/.zshrc.local
EOF
```

14.8.4 Paso 4: Configs Personales

```
# Usuario Alice
cat > users/alice/.zshrc.local << 'EOF'
# Alice's personal config

export EDITOR=nvim

alias mydata='cd /data/lab/alice'
EOF

# Usuario Bob
cat > users/bob/.zshrc.local << 'EOF'
# Bob's personal config

export EDITOR=vim
```

```
alias mydata='cd /data/lab/bob'
EOF
```

14.8.5 Paso 5: Script de Instalación

```
cat > install-lab.sh << 'EOF'
#!/bin/bash

USERNAME="$1"

if [ -z "$USERNAME" ]; then
    echo "Uso: $0 <username>"
    echo "Ejemplo: $0 alice"
    exit 1
fi

DOTFILES="$HOME/lab-dotfiles"

# Instalar común
cd "$DOTFILES/common"
stow git shell terminal

# Instalar personal del usuario
if [ -d "$DOTFILES/users/$USERNAME" ]; then
    cd "$DOTFILES/users/$USERNAME"
    stow .
    echo "  Configs de $USERNAME instaladas"
else
    echo "   No hay configs personales para $USERNAME"
fi

echo "  Instalación completa para $USERNAME"
EOF

chmod +x install-lab.sh
```

14.8.6 Paso 6: Cada Usuario Instala

```
# Usuario Alice:
cd ~/lab-dotfiles
./install-lab.sh alice

# Usuario Bob:
cd ~/lab-dotfiles
./install-lab.sh bob
```

14.8.7 Paso 7: Actualizar Configs Compartidas

```
# Cualquier usuario puede actualizar common/

# 1. Modificar
nano ~/lab-dotfiles/common/shell/.zshrc

# 2. Commit
cd ~/lab-dotfiles
git add common/shell/.zshrc
git commit -m "feat(shell): Add lab-wide utility function"

# 3. Push
git push origin main

# 4. Otros usuarios pull
git pull origin main
# Cambios se aplican automáticamente via symlinks
```

14.8.8 Paso 8: Usuarios Agregan Sus Configs

```
# Bob quiere agregar su config de Neovim

# 1. Crear su directorio personal
mkdir -p ~/lab-dotfiles/users/bob/.config

# 2. Copiar config
cp -r ~/.config/nvim ~/lab-dotfiles/users/bob/.config/

# 3. Commit (solo su carpeta)
cd ~/lab-dotfiles
git add users/bob/.config/nvim
git commit -m "feat(bob): Add Neovim configuration"
git push

# Otros usuarios no se afectan
```

14.9 Caso 9: Migrar de Sistema Manual a Stow

Escenario: Tienes dotfiles en GitHub pero SIN Stow (todos en raíz del repo). Quieres migrar a Stow.

14.9.1 Estado Inicial

```
# Tu repo actual (sin Stow):
~/dotfiles/
  .gitconfig
```

```
   .zshrc
   .zshenv
   nvim/
       init.lua
   konsolerc
   settings.json

# Estructura plana, difícil de gestionar
```

14.9.2 Paso 1: Backup Completo

```
# 1. Backup de dotfiles actuales
cp -r ~/dotfiles ~/dotfiles-backup-$(date +%Y%m%d)

# 2. Backup de HOME
mkdir -p ~/home-backup
cp ~/.zshrc ~/home-backup/
cp ~/.gitconfig ~/home-backup/
# etc...
```

14.9.3 Paso 2: Crear Nueva Estructura

```
cd ~/dotfiles

# Crear directorios de paquetes
mkdir -p git shell nvim terminal vscode
```

14.9.4 Paso 3: Reorganizar Archivos

```
cd ~/dotfiles

# Git
mv .gitconfig git/

# Shell
mv .zshrc shell/
mv .zshenv shell/

# Neovim (crear estructura correcta)
mkdir -p nvim/.config
mv nvim/ nvim/.config/nvim/

# Terminal
mkdir -p terminal/.config
mv konsolerc terminal/.config/
```

```
# VSCode
mkdir -p vscode/.config/Code/User
mv settings.json vscode/.config/Code/User/
```

14.9.5 Paso 4: Verificar Nueva Estructura

```
# Debería verse así:
tree -L 3 ~/dotfiles/

# ~/dotfiles/
#    git/
#        .gitconfig
#    shell/
#        .zshrc
#        .zshenv
#    nvim/
#        .config/
#            nvim/
#    terminal/
#        .config/
#            konsolerc
#    vscode/
#        .config/
#            Code/
#                User/
#                    settings.json
```

14.9.6 Paso 5: Eliminar Symlinks/Archivos Viejos de HOME

```
# Eliminar configs de HOME (los vamos a recrear con Stow)
rm ~/.gitconfig
rm ~/.zshrc
rm ~/.zshenv
rm -rf ~/.config/nvim
rm ~/.config/konsolerc
rm ~/.config/Code/User/settings.json
```

14.9.7 Paso 6: Instalar con Stow

```
cd ~/dotfiles

# Dry run primero
stow -nv git shell nvim terminal vscode

# Si todo OK, instalar
stow git shell nvim terminal vscode
```

```
# Verificar
ls -la ~/.gitconfig
ls -la ~/.zshrc
ls -la ~/.config/nvim
```

14.9.8 Paso 7: Commit Nueva Estructura

```
cd ~/dotfiles

# Stage todo
git add -A

# Ver cambios
git status

# Commit
git commit -m "refactor: Migrate to GNU Stow structure

BREAKING CHANGE: Repository structure changed to use Stow

- Organized configs into packages (git, shell, nvim, etc.)
- Each package replicates HOME directory structure
- Use 'stow <package>' to install

Migration guide:
1. stow -D * (if already installed)
2. stow git shell nvim terminal vscode"

# Push
git push origin main
```

14.9.9 Paso 8: Actualizar README

```
cat > README.md << 'EOF'
# Dotfiles (Stow-managed)

Personal configurations managed with GNU Stow.

# Structure

``
~/dotfiles/
   git/          # Git config
   shell/        # Zsh
   nvim/         # Neovim
   terminal/     # Konsole
```

```
   vscode/       # VSCode
``

## Installation

``bash
# Install Stow
sudo pacman -S stow

# Clone
git clone https://github.com/user/dotfiles.git ~/dotfiles

# Install all
cd ~/dotfiles
stow */

# Or selective
stow git shell nvim
``

## Update

``bash
cd ~/dotfiles
git pull
stow -R */
``

EOF

git add README.md
git commit -m "docs: Update README for Stow"
git push
```

14.9.10 Paso 9: Crear Scripts

```
# install.sh
cat > install.sh << 'EOF'
#!/bin/bash
cd "$HOME/dotfiles"
stow git shell nvim terminal vscode
echo "  Dotfiles installed"
EOF

chmod +x install.sh
git add install.sh
git commit -m "chore: Add install script"
```

```
git push
```

14.9.11 Paso 10: Limpiar Historial de Git (Opcional)

```
# Si tu repo era muy grande con historia antigua,
# puedes limpiarlo:

cd ~/dotfiles

# Crear orphan branch
git checkout --orphan latest_branch

# Add all files
git add -A

# Commit
git commit -m "refactor: Fresh start with Stow structure"

# Delete main
git branch -D main

# Rename current branch to main
git branch -m main

# Force push
git push -f origin main
```

14.10 Caso 10: Setup para Desarrollo Multi-Proyecto

Escenario: Trabajas en múltiples proyectos (Python, Web, Latex) y quieres configs específicas por proyecto.

14.10.1 Estructura de Dotfiles

```
~/dotfiles/
   common/           # Común a todo
       git/
       shell/
   python-dev/       # Python development
       nvim/
       vscode/
   web-dev/          # Web development
       nvim/
       vscode/
   latex-writing/    # Academic writing
       nvim/
       texstudio/
```

14.10.2 Paso 1: Crear Estructura

```
cd ~/dotfiles

# Común
mkdir -p common/{git,shell}

# Python dev
mkdir -p python-dev/{nvim,vscode}

# Web dev
mkdir -p web-dev/{nvim,vscode}

# LaTeX
mkdir -p latex-writing/{nvim,texstudio}
```

14.10.3 Paso 2: Configs Comunes

```
# Git (igual para todos)
cat > common/git/.gitconfig << 'EOF'
[user]
    name = Edison Achalma
    email = achalmaedison@gmail.com

[core]
    editor = nvim
EOF

# Shell base
cat > common/shell/.zshrc << 'EOF'
# Common shell config

# Aliases
alias gs='git status'
alias ll='ls -lah'

# Load project-specific config
[ -f ~/.zshrc.project ] && source ~/.zshrc.project
EOF
```

14.10.4 Paso 3: Configs Específicas por Proyecto

Python Development:

```
# Neovim para Python
cat > python-dev/nvim/.config/nvim/init.lua << 'EOF'
-- Python-focused Neovim config
```

```
-- LSP
require('lspconfig').pyright.setup{}

-- Python-specific keymaps
vim.keymap.set('n', '<leader>r', ':!python %<CR>')
EOF

# VSCode para Python
cat > python-dev/vscode/.config/Code/User/settings.json << 'EOF'
{
    "python.linting.enabled": true,
    "python.linting.pylintEnabled": true,
    "python.formatting.provider": "black"
}
EOF

# Shell additions para Python
cat > python-dev/shell/.zshrc.project << 'EOF'
# Python dev environment

export PYTHONPATH="$HOME/projects/python:$PYTHONPATH"

alias pytest='python -m pytest'
alias venv='python -m venv venv && source venv/bin/activate'
EOF
```

Web Development:

```
# Neovim para Web
cat > web-dev/nvim/.config/nvim/init.lua << 'EOF'
-- Web-focused Neovim config

-- LSP for JS/TS
require('lspconfig').tsserver.setup{}

-- Live server
vim.keymap.set('n', '<leader>l', ':!live-server .<CR>')
EOF

# VSCode para Web
cat > web-dev/vscode/.config/Code/User/settings.json << 'EOF'
{
    "emmet.includeLanguages": {
        "javascript": "javascriptreact"
    },
    "prettier.enable": true,
    "editor.formatOnSave": true
```

```
}
EOF
```

14.10.5 Paso 4: Scripts de Activación

```
# Script para activar proyecto Python
cat > ~/dotfiles/activate-python.sh << 'EOF'
#!/bin/bash

echo "  Activando entorno Python..."

cd ~/dotfiles

# Unstow otros proyectos
stow -D web-dev/nvim 2>/dev/null || true
stow -D latex-writing/nvim 2>/dev/null || true

# Stow común
stow common/*

# Stow Python
stow python-dev/*

# Copiar project-specific shell config
cp python-dev/shell/.zshrc.project ~/.zshrc.project

echo "  Entorno Python activado"
EOF

chmod +x ~/dotfiles/activate-python.sh
```

```
# Script para activar proyecto Web
cat > ~/dotfiles/activate-web.sh << 'EOF'
#!/bin/bash

echo "  Activando entorno Web..."

cd ~/dotfiles

# Unstow otros
stow -D python-dev/nvim 2>/dev/null || true
stow -D latex-writing/nvim 2>/dev/null || true

# Stow común
stow common/*

# Stow Web
```

```
stow web-dev/*

# Shell config
cp web-dev/shell/.zshrc.project ~/.zshrc.project

echo "  Entorno Web activado"
EOF

chmod +x ~/dotfiles/activate-web.sh
```

14.10.6 Paso 5: Uso

```
# Trabajar en proyecto Python
~/dotfiles/activate-python.sh
cd ~/projects/python/my-project
nvim  # Abre con config de Python

# Cambiar a proyecto Web
~/dotfiles/activate-web.sh
cd ~/projects/web/my-app
nvim  # Abre con config de Web
```

14.10.7 Paso 6: Automatizar con Direnv (Avanzado)

```
# Instalar direnv
sudo pacman -S direnv

# En cada proyecto, crear .envrc
cd ~/projects/python/my-project
cat > .envrc << 'EOF'
#!/bin/bash
# Activar entorno Python automáticamente
source "$HOME/dotfiles/activate-python.sh"
EOF

direnv allow

# Ahora al entrar al directorio, se activa automáticamente
```

15 Mi Repositorio .dotfiles

15.1 Mi Estructura Actual

```
~/dotfiles/
   git/
       .gitconfig
```

```
  kde/
      .config/
          kdeglobals
          dolphinrc
          ...
  shell/
      .zshrc
      starship.toml
  terminal/
      .config/
          konsolerc
  vscode/
      .config/
          settings.json
          keybindings.json
  zotero/
      .zotero/...
  obsidian/
      Documents/thoughts/.obsidian/
  ... (más paquetes)
```

15.2 Implementación de Stow

15.2.1 Script install.sh

Mi `install.sh` actual debe usar Stow. Aquí está mi versión mejorada:

```
#!/bin/bash
# ~/dotfiles/install.sh

set -e

DOTFILES="$HOME/dotfiles"
BACKUP_DIR="$HOME/dotfiles-backup-$(date +%Y%m%d-%H%M%S)"

# Colores
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

# Funciones
log_info() {
    echo -e "${GREEN}[INFO]${NC} $1"
}

log_warn() {
    echo -e "${YELLOW}[WARN]${NC} $1"
```

```
}

log_error() {
    echo -e "${RED}[ERROR]${NC} $1"
}

# Verificar que Stow está instalado
if ! command -v stow &> /dev/null; then
    log_error "Stow no está instalado"
    log_info "Instalando stow..."
    sudo apt update && sudo apt install -y stow
fi

# Función para hacer backup
backup_if_exists() {
    local file="$1"
    if [ -e "$file" ] && [ ! -L "$file" ]; then
        mkdir -p "$BACKUP_DIR"
        cp -r "$file" "$BACKUP_DIR/"
        log_warn "Backup: $file -> $BACKUP_DIR/"
    fi
}

# Función para stow paquete
stow_package() {
    local package="$1"

    log_info "Stowing $package..."

    # Dry run primero
    if stow -nv "$package" 2>&1 | grep -q "WARNING"; then
        log_warn "Conflicto detectado para $package"
        read -p "¿Hacer backup y continuar? [y/N] " -n 1 -r
        echo
        if [[ $REPLY =~ ^[Yy]$ ]]; then
            # Hacer backup de archivos conflictivos
            # (aquí necesitarías lógica más sofisticada)
            stow "$package"
        else
            log_error "Saltando $package"
            return 1
        fi
    else
        stow "$package"
        log_info "  $package instalado"
    fi
}
```

```
# Cambiar a dotfiles directory
cd "$DOTFILES" || exit 1

# Lista de paquetes a instalar
PACKAGES=(
    "git"
    "shell"
    "terminal"
    "kde"
    "vscode"
    "nvim"
    "kitty"
    # ... más paquetes
)

# Opción para instalar todo o selectivo
if [ "$1" == "all" ]; then
    PACKAGES=($(ls -d */ | sed 's#/#'))
    log_info "Instalando TODOS los paquetes"
elif [ $# -gt 0 ]; then
    PACKAGES=("$@")
    log_info "Instalando paquetes especificados: ${PACKAGES[*]}"
fi

# Instalar paquetes
for package in "${PACKAGES[@]}"; do
    stow_package "$package" || true
done

log_info "Instalación completa!"
if [ -d "$BACKUP_DIR" ]; then
    log_info "Backups guardados en: $BACKUP_DIR"
fi
```

Uso:

```
# Instalar paquetes específicos
./install.sh git shell terminal

# Instalar todo
./install.sh all

# Ver qué haría sin hacer cambios
# (modificar script para agregar -n flag)
```

15.2.2 Script para Desinstalar

```
#!/bin/bash
# ~/dotfiles/uninstall.sh

DOTFILES="$HOME/dotfiles"

cd "$DOTFILES" || exit 1

if [ $# -eq 0 ]; then
    echo "Uso: $0 <paquete1> [paquete2] ..."
    echo "O: $0 all"
    exit 1
fi

if [ "$1" == "all" ]; then
    PACKAGES=($(ls -d */ | sed 's#/#'))
else
    PACKAGES=("$@")
fi

for package in "${PACKAGES[@]}"; do
    echo "Unstowing $package..."
    stow -D "$package"
    echo "  $package desinstalado"
done
```

15.2.3 Reorganizar Paquetes Problemáticos

Zotero: Ubicación no estándar

```
# Actual:
zotero/
     .zotero/zotero/25vfdnq5.default/
         prefs.js

# Problema: .zotero está en HOME pero tiene subdirectorios profundos

# Solución 1: Usar como está (funciona)
stow zotero
# Resultado: ~/.zotero/... → dotfiles/zotero/.zotero/...

# Solución 2: Si solo quieres prefs.js, simplificar:
zotero/
     .zotero/
         zotero/
             25vfdnq5.default/
                 prefs.js
```

Obsidian: Ruta específica

```
# Actual:
obsidian/
    Documents/thoughts/.obsidian/

# Problema: No está en .config sino en Documents

# Solución: Está bien así, Stow lo maneja
stow obsidian
# Resultado: ~/Documents/thoughts/.obsidian → ...
```

15.2.4 .stowrc

Crear `~/dotfiles/.stowrc`:

```
# ~/dotfiles/.stowrc

# Target es siempre HOME
--target=$HOME

# Ignorar archivos comunes
--ignore='.git'
--ignore='README.*'
--ignore='LICENSE.*'
--ignore='.*.swp'
--ignore='.*~'
--ignore='install.sh'
--ignore='uninstall.sh'
--ignore='.stowrc'
```

Con esto, no necesitas especificar `-t ~` cada vez.

15.2.5 .stow-local-ignore por Paquete

Para vscode:

```
# ~/dotfiles/vscode/.stow-local-ignore

# No stow extensiones (solo configuración)
^/\.config/Code/CachedData/
^/\.config/Code/logs/
^/\.config/Code/User/workspaceStorage/
```

Para kde:

```
# ~/dotfiles/kde/.stow-local-ignore

# Archivos de sesión y cache
^/\.config/session/
^/\.cache/
```

Para shell:

```
# ~/dotfiles/shell/.stow-local-ignore

# Historia de shells (puede tener info sensible)
^/\.zsh_history
^/\.bash_history

# Archivos compilados
\.zcompdump
```

15.2.6 Script de Verificación

```
#!/bin/bash
# ~/dotfiles/check-stow.sh

# Verificar qué está stowed

DOTFILES="$HOME/dotfiles"

echo "Paquetes stowed:"
echo "================"

cd "$DOTFILES" || exit 1

for package in */; do
    package=${package%/}

    # Encontrar primer archivo del paquete
    first_file=$(find "$package" -type f | head -1)

    if [ -z "$first_file" ]; then
        continue
    fi

    # Convertir a path en HOME
    home_path="$HOME/${first_file#$package/}"

    if [ -L "$home_path" ]; then
        target=$(readlink "$home_path")
        if [[ "$target" == *"$DOTFILES/$package"* ]]; then
            echo "  $package"
        else
            echo "  $package (symlink apunta a otro lugar)"
        fi
    else
        echo "  $package (no stowed)"
```

```
    fi
done
```

15.2.7 Actualizar .gitignore

```
# ~/dotfiles/.gitignore

# Backups
*~
*.bak
*.old
*.orig
.*.swp

# Datos sensibles
shell/.zsh_history
shell/.bash_history
.netrc
.authinfo

# Cache y temporales
**/.cache/
**/__pycache__/
**/node_modules/

# Logs
**/*.log

# Sistema
.DS_Store
Thumbs.db

# Archivos de Stow
.stow

# Zotero database (demasiado grande)
zotero/.zotero/zotero/*/zotero.sqlite*

# VSCode workspace storage
vscode/.config/Code/User/workspaceStorage/
```

15.2.8 Comandos Útiles

```
# Navegar a dotfiles
cd ~/dotfiles

# Instalar todo (primera vez)
```

```
./install.sh all

# Instalar paquetes esenciales
./install.sh git shell terminal kde

# Verificar qué está instalado
./check-stow.sh

# Actualizar después de pull
git pull
stow -R */  # Restow todo

# Desinstalar temporalmente para pruebas
stow -D vscode
# hacer pruebas...
stow vscode  # Reinstalar

# Agregar nuevo paquete
mkdir new-app
# crear estructura...
stow new-app
git add new-app/
git commit -m "Add new-app"
```

16 Workflows

16.1 Workflow 1: Configuración Inicial

```
# Paso 1: Crear estructura
mkdir -p ~/dotfiles
cd ~/dotfiles
git init

# Paso 2: Crear paquetes
mkdir -p zsh nvim git

# Paso 3: Mover configs existentes
mv ~/.zshrc zsh/
mv ~/.config/nvim nvim/.config/
mv ~/.gitconfig git/

# Paso 4: Stow
stow zsh nvim git

# Paso 5: Verificar
ls -la ~/.zshrc  # debe ser symlink
```

```
# Paso 6: Git
git add .
git commit -m "Initial dotfiles"
git remote add origin git@github.com:user/dotfiles.git
git push -u origin main
```

16.2 Workflow 2: Día a Día

```
# Editar configuración (desde cualquier lugar)
nvim ~/.config/nvim/init.lua  # Edita a través del symlink

# Commit cambios
cd ~/dotfiles
git add nvim/
git commit -m "Update nvim config: add new plugin"
git push

# En otra máquina
cd ~/dotfiles
git pull
# Los cambios se reflejan automáticamente (symlinks)
```

16.3 Workflow 3: Nueva Máquina

```
# Clonar
git clone https://github.com/user/dotfiles.git ~/dotfiles

# Instalar Stow
sudo apt install stow

# Backup existentes (precaución)
mkdir ~/backup
cp ~/.zshrc ~/backup/ 2>/dev/null || true

# Stow
cd ~/dotfiles
stow */

# Verificar
ls -la ~/ | grep '\->'

# Instalar dependencias de apps
# (nvim plugins, zsh plugins, etc)
```

16.4 Workflow 4: Experimentar

```
# Crear branch de experimento
cd ~/dotfiles
git checkout -b experiment-new-nvim

# Modificar libremente
nvim nvim/.config/nvim/init.lua

# Restow para aplicar
stow -R nvim

# Probar...

# Si funciona:
git checkout main
git merge experiment-new-nvim

# Si no funciona:
git checkout main
stow -R nvim  # Vuelve a main automáticamente
```

16.5 Workflow 5: Actualización Limpia

```
# Pull cambios
cd ~/dotfiles
git pull origin main

# Verificar qué cambió
git log -p --since="1 week ago"

# Desinstalar y reinstalar (limpia symlinks obsoletos)
stow -D nvim
stow nvim

# O usar restow
stow -R nvim

# Verificar que funciona
nvim --version
```

17 Best Practices

17.1 Organización de Paquetes

DO:

```
# Un paquete por aplicación
~/dotfiles/
   nvim/
   zsh/
   git/

# Replicar estructura de HOME exactamente
nvim/
     .config/
         nvim/
             init.lua
```

DON'T:

```
# Múltiples aplicaciones en un paquete
~/dotfiles/
   configs/
       .config/nvim/
       .config/kitty/
       .zshrc

# Estructura diferente a HOME
nvim/
     init.lua  #  Falta .config/nvim/
```

17.2 Uso de Ignore Lists

DO:

```
# Ignorar archivos sensibles
# .stow-global-ignore
**/.history
**/.ssh/id_*
.netrc

# Ignorar cache por paquete
# nvim/.stow-local-ignore
^/\.config/nvim/plugin/packer_compiled\.lua
```

DON'T:

```
# Commit archivos sensibles sin ignorar
git add ~/.ssh/id_rsa  #  ¡NUNCA!
```

17.3 Commits y Mensajes

DO:

```
# Commits descriptivos y atómicos
git commit -m "feat(nvim): Add LSP configuration for Rust"
git commit -m "fix(zsh): Correct path to starship prompt"

# Un cambio lógico por commit
```

DON'T:

```
# Commits genéricos
git commit -m "Update stuff"
git commit -m "Changes"

# Múltiples cambios no relacionados en un commit
```

17.4 Testing Antes de Commit

DO:

```
# Siempre test antes de commit
stow -D nvim  # Desinstalar
stow nvim     # Reinstalar
# Verificar que funciona
git commit

# Dry run en nueva máquina
stow -nv */
```

DON'T:

```
# Commit sin probar
# Cambios → commit → push → Rompe en otra máquina
```

17.5 Backup Siempre

DO:

```
# Backup antes de stow en nueva máquina
mkdir ~/backup
cp -r ~/.config/nvim ~/backup/

# Luego stow
stow nvim
```

DON'T:

```
# Stow directamente sin backup
stow nvim  #  Puede sobrescribir configs importantes
```

17.6 Documentación

DO:

```
# README.md completo
# - Qué paquetes hay
# - Cómo instalar
# - Dependencias
# - Comandos útiles

# Comentarios en configs
# nvim/init.lua
-- LSP configuration
-- Requires: nvim-lspconfig plugin
```

DON'T:

```
# README vacío o sin info
# Configs sin comentarios
```

17.7 Estructura Consistente

DO:

```
# Misma estructura en todos los paquetes
package/
    .stow-local-ignore
    README.md
    (archivos que van en HOME)
```

DON'T:

```
# Estructura inconsistente entre paquetes
```

17.8 Versionado

DO:

```
# Tags para versiones estables
git tag -a v1.0.0 -m "Stable nvim config"

# Branches para experimentar
git checkout -b experiment/new-theme
```

DON'T:

```
# Todo en main sin tags
# Experimentar directamente en main
```

18 Alternativas a Stow

18.1 Yadm (Yet Another Dotfiles Manager)

Ventajas:

- Git nativo, no symlinks
- Encriptación built-in
- Templates con Jinja2
- Bootstrap scripts

Desventajas:

- Menos control granular
- Todo en un repo

```
# Instalar
sudo apt install yadm

# Usar
yadm init
yadm add ~/.zshrc
yadm commit -m "Add zshrc"
```

18.2 Chezmoi

Ventajas:

- Templates
- Secrets management
- Cross-platform
- Estado vs archivos

Desventajas:

- Más complejo
- Curva de aprendizaje

```
# Instalar
sh -c "$(curl -fsLS get.chezmoi.io)"

# Usar
chezmoi init
chezmoi add ~/.zshrc
```

18.3 Dotbot

Ventajas:

- Basado en configuración YAML
- Bootstrapping automático
- Plugins

Desventajas:

- Otra herramienta que aprender
- Menos flexibilidad que Stow

```
# install.conf.yaml
- link:
    ~/.zshrc: zshrc
    ~/.config/nvim: nvim
```

18.4 Bare Git Repository

Ventajas:

- Solo Git, no tools extra
- Total control

Desventajas:

- Más manual
- Conflictos con .gitignore

```
# Setup
git init --bare $HOME/.dotfiles
alias config='/usr/bin/git --git-dir=$HOME/.dotfiles/ --work-tree=$HOME'
config config --local status.showUntrackedFiles no

# Usar
config add .zshrc
config commit -m "Add zshrc"
```

18.5 Comparación

Característica	GNU Stow	yadm	chezmoi	dotbot	Repositorio Git bare
Simplicidad	Muy alta	Alta	Media	Alta	Media
Flexibilidad	Muy alta	Media	Muy alta	Media	Muy alta
Soporte para plantillas	No	Parcial	Completo (Jinja2)	No	No

Característica	GNU Stow	yadm	chezmoi	dotbot	Repositorio Git bare
Manejo de secretos	No	Bueno	Excelente (integrado)	No	No
Facilidad de instalación	Muy sencilla	Muy sencilla	Muy sencilla	Muy sencilla	Sin instalación adicional
Tamaño de la comunidad	Grande	Mediana	Grande	Pequeña	N/A (herramienta nativa)
Curva de aprendizaje	Baja	Baja	Media-alta	Baja	Media
Uso de enlaces simbólicos	Sí (principal)	No	Sí (opcional)	Sí	No
Soporte multiplataforma	Excelente	Bueno	Excelente	Bueno	Excelente

Recomendación: Stow es ideal si quieres:

- Simplicidad
- Control total
- Organización por paquetes
- Solo symlinks, sin magia

19 Conclusión

La gestión de dotfiles es una práctica esencial para optimizar el entorno de desarrollo y asegurar la persistencia de las configuraciones personalizadas. GNU Stow, en particular, se destaca por su simplicidad y eficacia al manejar enlaces simbólicos, especialmente cuando se combina con Git para el versionado y la sincronización. Permite una modularidad excelente y una replicación rápida de entornos.

Si bien existen alternativas más avanzadas como Chezmoi o YADM (que ofrecen funciones adicionales como plantillas y cifrado de secretos) o soluciones declarativas como NixOS/Home-Manager, Stow sigue siendo una opción robusta y preferida por muchos por su enfoque directo y la curva de aprendizaje mínima. La clave es elegir la herramienta que mejor se adapte a las necesidades y al nivel de complejidad deseado, siempre priorizando la seguridad de la información sensible.

19.1 Comandos Esenciales

```
# Instalar
stow paquete

# Desinstalar
stow -D paquete

# Reinstalar
stow -R paquete
```

```
# Simular
stow -nv paquete

# Ver qué hace
stow -vv paquete

# Ignorar archivos
stow --ignore='patrón' paquete

# Especificar directorios
stow -d ~/dotfiles -t ~ paquete
```

19.2 Recursos Adicionales

Documentación:

- Manual oficial: `man stow`
- Info pages: `info stow`
- Web: https://www.gnu.org/software/stow/

Comunidad:

- r/unixporn (ejemplos de dotfiles)
- GitHub topic: dotfiles
- YouTube: “dotfiles management”

Ejemplos de dotfiles con Stow:

- https://github.com/search?q=stow+dotfiles
- https://dotfiles.github.io/

20 Publicaciones Similares

Si te interesó este artículo, te recomendamos que explores otros blogs y recursos relacionados que pueden ampliar tus conocimientos. Aquí te dejo algunas sugerencias:

1. Comandos De Informacion Windows
2. Adb
3. Limpieza Y Optimizacion De Pc
4. Usando Apk En Windown 11
5. Gestionar Versiones De Jdk En Kubuntu
6. Instalar Tor Browser
7. Crear Enlaces Duros O Hard Link En Linux
8. Comandos Vim
9. Guia De Git Y Github
10. 00 Primeros Pasos En Linux
11. 01 Introduccion Linux
12. 02 Distribuciones Linux
13. 03 Instalacion Linux
14. 04 Administracion Particiones Volumenes

15. Atajos De Teclado Y Comandos Para Usar Vim
16. Instalando Specitify
17. Gestiona Tus Dotfiles Con Gnu Stow

Esperamos que encuentres estas publicaciones igualmente interesantes y útiles. ¡Disfruta de la lectura!