

Complete ASDF DAT Installation & Deployment Guide

A comprehensive step-by-step tutorial for setting up, deploying, and running the ASDF DAT automated buyback and burn system on Solana.

Table of Contents

1. [Prerequisites Installation](#)
 2. [Project Setup](#)
 3. [Wallet Configuration](#)
 4. [Program Deployment](#)
 5. [Bot Configuration](#)
 6. [Running the System](#)
 7. [Production Deployment](#)
 8. [Monitoring & Maintenance](#)
 9. [Troubleshooting](#)
-

Prerequisites Installation

Step 1: Install Node.js (v18+)

Windows:

```
powershell

# Download from https://nodejs.org/
# Choose LTS version (18.x or higher)
# Run installer and follow prompts

# Verify installation
node --version
npm --version
```

macOS:

```
bash
```

```
# Using Homebrew  
brew install node@18
```

```
# Or download from https://nodejs.org/  
# Verify installation  
node --version  
npm --version
```

Linux (Ubuntu/Debian):

```
bash  
  
# Add NodeSource repository  
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -  
  
# Install Node.js  
sudo apt-get install -y nodejs  
  
# Verify installation  
node --version  
npm --version
```

Step 2: Install Rust

All Platforms:

```
bash  
  
# Install Rust using rustup  
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh  
  
# Follow the prompts (press 1 for default installation)  
# Restart terminal or run:  
source $HOME/.cargo/env  
  
# Verify installation  
rustc --version  
cargo --version
```

Step 3: Install Solana CLI

Windows:

```
powershell
```

```
# Open PowerShell as Administrator
curl https://release.solana.com/stable/solana-install-init-x86_64-pc-windows-msvc.exe -OutFile C:\solana-install-tmp\sc

# Run installer
C:\solana-install-tmp\solana-install-init.exe stable

# Add to PATH manually if needed
# Restart PowerShell and verify
solana --version
```

macOS/Linux:

```
bash

# Install Solana CLI
sh -c "$(curl -sSfL https://release.solana.com/stable/install)"

# Add to PATH
export PATH="$HOME/.local/share/solana/install/active_release/bin:$PATH"

# Add to shell profile (.bashrc, .zshrc, etc.)
echo 'export PATH="$HOME/.local/share/solana/install/active_release/bin:$PATH"' >> ~/.bashrc

# Verify installation
solana --version
```

Step 4: Install Anchor Framework

```
bash

# Install Anchor Version Manager (AVM)
cargo install --git https://github.com/coral-xyz/anchor avm --locked --force

# Install Anchor 0.30.0
avm install 0.30.0
avm use 0.30.0

# Verify installation
anchor --version
```

Project Setup

Step 1: Clone or Create Project Directory

```
bash

# Create project directory
mkdir asdf-dat
cd asdf-dat

# Initialize Anchor project (if starting fresh)
anchor init asdf-dat --javascript
cd asdf-dat

# Or clone from repository (if available)
git clone https://github.com/yourusername/asdf-dat.git
cd asdf-dat
```

Step 2: Install Dependencies

```
bash

# Install Node.js packages
npm install

# Install additional required packages
npm install @solana/web3.js @solana/spl-token @coral-xyz/anchor
npm install dotenv winston node-cron axios express
npm install --save-dev typescript @types/node ts-node
```

Step 3: Project Structure Setup

```
bash
```

```
# Create necessary directories
```

```
mkdir -p logs
```

```
mkdir -p backups
```

```
mkdir -p scripts
```

```
mkdir -p dist
```

```
# Create project files (copy from provided files)
```

```
# Place the provided files in their respective locations:
```

```
# - src/bot.ts
```

```
# - src/config.ts
```

```
# - src/utils.ts
```

```
# - src/types.ts
```

```
# - programs/asdf-dat/src/lib.rs
```

```
# - Anchor.toml
```

```
# - package.json
```

```
# - .env.example
```

Wallet Configuration

Step 1: Setup CTO Wallet

Since you already have the CTO wallet that can access PumpSwap fees:

Option A: Export from Phantom/Solflare

```
bash
```

```
# If using Phantom:
```

```
# 1. Settings → Security & Privacy → Show Private Key
```

```
# 2. Copy the private key (base58 format)
```

```
# Create wallet.json from private key
```

```
solana-keygen recover -o wallet.json
```

```
# Paste your private key when prompted
```

Option B: Use Existing Solana Keypair File

```
bash
```

```
# Copy your existing keypair file
```

```
cp /path/to/your/cto-keypair.json wallet.json
```

Step 2: Verify Wallet Address

```
bash
```

```
# Check that it matches your CTO wallet
```

```
solana address -k wallet.json
```

```
# Should output: vcGYZbvDid6cRUkCCqcWpBxow73TLpmY6ipmDUtrTF8
```

Step 3: Check Wallet Balance

```
bash
```

```
# Configure Solana for mainnet
```

```
solana config set --url https://api.mainnet-beta.solana.com
```

```
# Check balance
```

```
solana balance wallet.json
```

```
# Ensure you have at least 2-3 SOL for deployment
```

Program Deployment

Step 1: Configure Anchor.toml

Edit `Anchor.toml`:

```
toml
```

```
[features]
```

```
seeds = false
```

```
skip-lint = false
```

```
[programs.mainnet]
```

```
asdf_dat = "ASDFdatBuybackBurnXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

```
[provider]
```

```
cluster = "mainnet"
```

```
wallet = "./wallet.json"
```

```
[cluster.mainnet]
```

```
url = "https://api.mainnet-beta.solana.com"
```

Step 2: Build the Program

```
bash
```

```
# Build the Solana program
```

```
anchor build
```

```
# This will:
```

```
# - Compile the Rust program
```

```
# - Generate IDL file
```

```
# - Create deployment artifacts
```

Step 3: Deploy to Devnet (Testing)

```
bash
```

```
# Switch to devnet for testing
```

```
solana config set --url https://api.devnet.solana.com
```

```
# Airdrop SOL for testing (devnet only)
```

```
solana airdrop 2 wallet.json
```

```
# Deploy to devnet
```

```
anchor deploy --provider.cluster devnet
```

```
# Note the Program ID displayed
```

```
# Example: Program Id: 7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU
```

Step 4: Deploy to Mainnet (Production)

```
bash
```

```
# Switch to mainnet
```

```
solana config set --url https://api.mainnet-beta.solana.com
```

```
# Ensure wallet has sufficient SOL (2-3 SOL recommended)
```

```
solana balance wallet.json
```

```
# Deploy to mainnet
```

```
anchor deploy
```

```
# Save the Program ID - IMPORTANT!
```

```
# Example output:
```

```
# Program Id: 7xKXtg2CW87d97TXJSDpbD5jBkheTqA83TZRuJosgAsU
```

Step 5: Update Program ID

Update the Program ID in multiple files:

1. In `.env`:

```
env
```

```
PROGRAM_ID=YOUR_DEPLOYED_PROGRAM_ID_HERE
```

2. In `Anchor.toml`:

```
toml
```

```
[programs.mainnet]  
asdf_dat = "YOUR_DEPLOYED_PROGRAM_ID_HERE"
```

3. In `lib.rs`:

```
rust
```

```
declare_id!("YOUR_DEPLOYED_PROGRAM_ID_HERE");
```

4. Rebuild after updating lib.rs:

```
bash
```

```
anchor build
```

Bot Configuration

Step 1: Environment Setup

Create `.env` file from template:

```
bash
```

```
cp .env.example .env
```

Edit `.env`:

```
env
```



```
# Wallet Configuration
WALLET_PATH=./wallet.json

# Program Configuration (use your deployed ID)
PROGRAM_ID=YOUR_DEPLOYED_PROGRAM_ID_HERE

# Network Configuration
NETWORK=mainnet
RPC_URL=https://api.mainnet-beta.solana.com

# Logging
LOG_LEVEL=info
LOG_FILE=./logs/dat-bot.log
DEBUG=false

# Performance Tuning
MAX_RETRIES=3
RETRY_DELAY=1000
COMMITMENT_LEVEL=confirmed
```

Step 2: Compile TypeScript

```
bash

# Compile TypeScript files
npm run compile

# This creates JavaScript files in dist/
```

Step 3: Initialize the DAT

```
bash

# Initialize the on-chain DAT state (only once!)
npm run dat:init

# Expected output:
# DAT initialized successfully
# Authority: vcGYZbvDid6cRUkCCqcWpBxow73TLpmY6ipmDUtrTF8
# CTO Wallet: vcGYZbvDid6cRUkCCqcWpBxow73TLpmY6ipmDUtrTF8
```

Running the System

Step 1: Test Operations

```
bash

# Check available fees in creator vault
npm run dat:check

# Execute a manual cycle (if fees available)
npm run dat:cycle

# View statistics
npm run dat:stats
```

Step 2: Run Automated Bot

```
bash

# Start the automated bot (runs every 6 hours)
npm run dat:bot

# Output:
# ASDF DAT BOT - AUTOMATED MODE
# Configuration:
# - Check Interval: 6 hours
# - Minimum Fees: 0.05 SOL
# - Cycle Times: 0:00, 6:00, 12:00, 18:00 UTC
```

Production Deployment

Step 1: Install PM2

```
bash

# Install PM2 globally
npm install -g pm2

# Install PM2 log rotation
pm2 install pm2-logrotate
```

Step 2: Start with PM2

```
bash
```

```
# Start the bot with PM2
pm2 start ecosystem.config.js

# Or directly:
pm2 start npm --name "asdf-dat" -- run dat:bot

# Save PM2 configuration
pm2 save

# Setup auto-restart on system reboot
pm2 startup

# Follow the command it outputs
```

Step 3: Create Ecosystem File

Create `ecosystem.config.js`:

```
javascript

module.exports = {
  apps: [{
    name: 'asdf-dat',
    script: 'npm',
    args: 'run dat:bot',
    instances: 1,
    autorestart: true,
    watch: false,
    max_memory_restart: '1G',
    env: {
      NODE_ENV: 'production'
    },
    error_file: './logs/pm2-error.log',
    out_file: './logs/pm2-out.log',
    log_file: './logs/pm2-combined.log',
    time: true
  }]
};
```

Step 4: Setup Monitoring

```
bash
```

```
# Run monitoring dashboard
chmod +x scripts/monitor.sh
./scripts/monitor.sh

# Or use PM2 monitoring
pm2 monit
```

Monitoring & Maintenance

Daily Tasks

```
bash

# Check bot status
pm2 status asdf-dat

# View recent logs
pm2 logs asdf-dat --lines 100

# Check statistics
npm run dat:stats

# Monitor wallet balance
solana balance wallet.json
```

Weekly Tasks

```
bash

# Check for errors in logs
grep ERROR logs/dat-bot.log | tail -20

# Verify burn statistics
npm run dat:stats

# Backup logs
tar -czf backups/logs-$(date +%Y%m%d).tar.gz logs/
```

Emergency Controls

```
bash
```

```
# Pause operations
```

```
npm run dat:pause
```

```
# Resume operations
```

```
npm run dat:resume
```

```
# Restart bot
```

```
pm2 restart asdf-dat
```

```
# Stop bot
```

```
pm2 stop asdf-dat
```

Troubleshooting

Common Issues and Solutions

Issue: "Insufficient fees in vault"

```
bash
```

```
# Check creator vault balance
```

```
npm run dat:check
```

```
# Solution: Wait for fees to accumulate above 0.05 SOL
```

Issue: "Transaction failed"

```
bash
```

```
# Check wallet balance
```

```
solana balance wallet.json
```

```
# Ensure sufficient SOL for gas (minimum 0.1 SOL)
```

```
# Check RPC status
```

```
curl https://api.mainnet-beta.solana.com/health
```

Issue: "Program not found"

```
bash
```

```
# Verify program deployment
solana program show YOUR_PROGRAM_ID

# Redeploy if necessary
anchor deploy
```

Issue: "Bot not starting"

```
bash

# Check Node version
node --version # Must be 18+

# Rebuild dependencies
rm -rf node_modules
npm install
npm run compile
```

Issue: "Anchor build fails"

```
bash

# Clear build cache
anchor clean

# Update Rust
rustup update

# Rebuild
anchor build
```

Issue: "Permission denied on wallet.json"

```
bash

# Fix permissions (Linux/Mac)
chmod 600 wallet.json

# Windows: Right-click → Properties → Security
# Set read/write for current user only
```

Log Files Location

```
bash
```

```
# Application logs
tail -f logs/dat-bot.log
```

```
# PM2 logs
pm2 logs asdf-dat
```

```
# System logs (if using systemd)
journalctl -u asdf-dat -f
```

Performance Optimization

```
bash

# Use a premium RPC endpoint for better performance
# Update .env:
RPC_URL=https://your-premium-rpc.com

# Adjust gas settings in .env:
MAX_PRIORITY_FEE=2000
COMPUTE_UNIT_LIMIT=300000
```

Security Best Practices

1. Wallet Security:

- Never commit `wallet.json` to Git
- Use hardware wallet for production
- Implement multi-signature if possible
- Keep backup of seed phrase in secure location

2. Access Control:

- Limit server access (SSH keys only)
- Use firewall rules (ufw/iptables)
- Enable 2FA on all accounts
- Regular security audits

3. Monitoring:

- Set up alerts for failures
- Monitor wallet balance
- Track burn statistics
- Watch for unusual activity

4. Backups:

- Regular backup of logs
 - Backup wallet securely (encrypted)
 - Document all configurations
 - Version control for code
-

Advanced Configuration

Using Hardware Wallet (Ledger)

```
bash

# Install Ledger support
npm install @ledgerhq/hw-transport-node-hid
npm install @solana/wallet-adapter-ledger

# Configure for Ledger (requires custom implementation)
```

Setting Up Alerts

Discord Webhook:

```
env

# In .env
ALERT_WEBHOOK_URL=https://discord.com/api/webhooks/YOUR_WEBHOOK
```

Email Alerts:

```
env

# In .env
ALERT_EMAIL=admin@yourdomain.com
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASS=your-app-password
```

Custom RPC Endpoints

Helius:

```
env
```



```
RPC_URL=https://mainnet.helius-rpc.com/?api-key=YOUR_KEY
```

QuickNode:

```
env
```

```
RPC_URL=https://YOUR-ENDPOINT.quiknode.pro/YOUR_KEY/
```

Alchemy:

```
env
```

```
RPC_URL=https://solana-mainnet.g.alchemy.com/v2/YOUR_KEY
```

Verification Checklist

Pre-Deployment

- ☐ Node.js v18+ installed
- ☐ Rust and Cargo installed
- ☐ Solana CLI configured
- ☐ Anchor 0.30.0 installed
- ☐ All dependencies installed
- ☐ Project structure created

Deployment

- ☐ Wallet created and secured
- ☐ Wallet funded (2-3 SOL)
- ☐ Program compiled successfully
- ☐ Program deployed to mainnet
- ☐ Program ID updated in all files
- ☐ Environment variables configured

Post-Deployment

- ☐ DAT initialized on-chain
- ☐ Test cycle executed successfully
- ☐ Bot running in automated mode
- ☐ PM2 configured for production
- ☐ Monitoring dashboard working
- ☐ Logs being generated correctly
- ☐ Backup system in place

Production Readiness

- ☐ Security measures implemented
 - ☐ Alert system configured
 - ☐ Documentation complete
 - ☐ Team access configured
 - ☐ Recovery procedures documented
 - ☐ Performance optimized
-

Support Resources

- **Solana Documentation:** <https://docs.solana.com>
 - **Anchor Documentation:** <https://www.anchor-lang.com>
 - **Node.js Documentation:** <https://nodejs.org/docs>
 - **PM2 Documentation:** <https://pm2.keymetrics.io>
 - **Discord Community:** [Your Discord Link]
 - **GitHub Issues:** [Your GitHub Repo]/issues
 - **Emergency Contact:** [Your Contact Info]
-

Quick Reference Commands

```
bash
```

Development

```
npm run dat:check    # Check fees
npm run dat:cycle    # Manual cycle
npm run dat:stats    # View stats
npm run dat:bot      # Start bot
```

Production

```
pm2 start asdf-dat   # Start with PM2
pm2 stop asdf-dat    # Stop bot
pm2 restart asdf-dat # Restart bot
pm2 logs asdf-dat    # View logs
pm2 monit            # Monitor
```

Maintenance

```
npm run dat:pause    # Emergency pause
npm run dat:resume    # Resume operations
anchor deploy        # Redeploy program
solana balance wallet.json # Check balance
```

Appendix: Complete File List

```
asdf-dat/
├── programs/
│   └── asdf-dat/
│       ├── src/
│       │   └── lib.rs
│       ├── Cargo.toml
│       └── Xargo.toml
├── src/
│   ├── bot.ts
│   ├── config.ts
│   ├── utils.ts
│   └── types.ts
├── scripts/
│   ├── install.sh
│   └── monitor.sh
├── tests/
│   └── asdf-dat.test.ts
├── logs/
│   └── (generated logs)
├── dist/
│   └── (compiled JS)
├── target/
│   └── (build artifacts)
```



- └─ .env
- └─ .env.example
- └─ .gitignore
- └─ Anchor.toml
- └─ Cargo.toml
- └─ package.json
- └─ tsconfig.json
- └─ ecosystem.config.js
- └─ wallet.json
- └─ README.md

Congratulations! Your ASDF DAT automated buyback and burn system is now fully deployed and operational. The bot will automatically execute cycles every 6 hours when sufficient fees are available in the creator vault.

For additional support or questions, please refer to the community resources listed above.