Jenkins Cheatsheet

Essential commands for automation, CI/CD, and deployment pipelines

This cheatsheet provides a quick reference to fundamental Jenkins operations, commands, and best practices, ideal for both beginners and experienced developers for efficient continuous integration and deployment automation.

Job Management
Create, configure, and control build jobs

Pipeline Operations
Build and manage CI/CD pipelines

System Monitoring
Monitor builds and troubleshoot issues

Plugin Management
Install and configure Jenkins plugins

User & Security
Manage users, roles, and credentials

Installation & Setup

Linux Installation

Install Jenkins on Ubuntu/Debian systems.

```
# Update package manager and install Java
sudo apt update
sudo apt install fontconfig openjdk-21-jre
java -version

# Add Jenkins GPG key
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# Add Jenkins repository
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

# Install Jenkins
sudo apt update && sudo apt install jenkins

# Start Jenkins service
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Windows & macOS

Install Jenkins using installers or package managers.

```
# Windows: Download Jenkins installer from jenkins.io
# Or use Chocolatey
choco install jenkins

# macOS: Use Homebrew
brew install jenkins-lts
# Or download directly from:
# https://www.jenkins.io/download/

# Start Jenkins service
brew services start jenkins-lts
```

Post-Installation Setup

Initial configuration and unlock Jenkins.

```
# Get initial admin password
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

# Or for Docker installations
docker exec jenkins_container cat /var/jenkins_home/secrets/initialAdminPassword

# Access Jenkins web interface
# Browse to http://localhost:8080
# Enter the initial admin password
# Install suggested plugins or select custom plugins
```

Initial Configuration

Complete the setup wizard and create admin user.

```
# After unlocking Jenkins:
# 1. Install suggested plugins (recommended)
# 2. Create first admin user
# 3. Configure Jenkins URL
# 4. Start using Jenkins

# Verify Jenkins is running
sudo systemctl status jenkins
# Check Jenkins logs if needed
sudo journalctl -u jenkins.service
```

Basic Jenkins Operations

Essential operations to get started with Jenkins automation and build management.

01

Access Jenkins: Web Interface & CLI Setup

Access Jenkins through browser and set up CLI tools.

```
# Access Jenkins web interface
http://localhost:8080

# Download Jenkins CLI
wget http://localhost:8080/jnlpJars/jenkins-cli.jar

# Test CLI connection
java -jar jenkins-cli.jar -s http://localhost:8080 help

# List available commands
java -jar jenkins-cli.jar -s http://localhost:8080 help
```

02

Job Creation: `create-job` / Web UI

Create new build jobs using CLI or web interface.

```
# Create job from XML configuration
java -jar jenkins-cli.jar -auth user:token create-job my-job < job-config.xml

# Create simple freestyle job via web UI:
# 1. Click "New Item"
# 2. Enter job name
# 3. Select "Freestyle project"
# 4. Configure build steps
# 5. Save configuration
```

03

List Jobs: `list-jobs`

View all configured jobs in Jenkins.

```
# List all jobs
java -jar jenkins-cli.jar -auth user:token list-jobs

# List jobs with pattern matching
java -jar jenkins-cli.jar -auth user:token list-jobs "*test*"

# Get job configuration
java -jar jenkins-cli.jar -auth user:token get-job my-job > job-config.xml
```

Job Management

Build Jobs: `build`

Trigger and manage job builds.

```
# Build a job
java -jar jenkins-cli.jar -auth user:token build my-job

# Build with parameters
java -jar jenkins-cli.jar -auth user:token build my-job -p PARAM=value

# Build and wait for completion
java -jar jenkins-cli.jar -auth user:token build my-job -s -v

# Build and follow console output
java -jar jenkins-cli.jar -auth user:token build my-job -f
```

Job Control: `enable-job` / `disable-job`

Enable or disable jobs.

```
# Enable a job
java -jar jenkins-cli.jar -auth user:token enable-job my-job

# Disable a job
java -jar jenkins-cli.jar -auth user:token disable-job my-job

# Check job status in web UI
# Navigate to job dashboard
# Look for "Disable/Enable" button
```

Job Deletion: `delete-job`

Remove jobs from Jenkins.

```
# Delete a job
java -jar jenkins-cli.jar -auth user:token delete-job my-job

# Bulk delete jobs (with caution)
for job in job1 job2 job3; do
  java -jar jenkins-cli.jar -auth user:token delete-job $job
done
```

Console Output: `console`

View build logs and console output.

```
# View latest build console output
java -jar jenkins-cli.jar -auth user:token console my-job

# View specific build number
java -jar jenkins-cli.jar -auth user:token console my-job 15

# Follow console output in real-time
java -jar jenkins-cli.jar -auth user:token console my-job -f
```

Pipeline Management

Pipeline Creation

Create and configure Jenkins pipelines.

```
# Basic Jenkinsfile (Declarative Pipeline)
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building application...'
        sh 'make build'
      }
    }

    stage('Test') {
      steps {
        echo 'Running tests...'
        sh 'make test'
      }
    }

    stage('Deploy') {
      steps {
        echo 'Deploying application...'
        sh 'make deploy'
      }
    }
  }
```

Pipeline Configuration

Advanced pipeline configuration and options.

```
# Pipeline with post-build actions
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        sh 'make build'
      }
    }
  }

  post {
    always {
      echo 'This always runs'
    }
    success {
      echo 'Build succeeded'
    }
    failure {
      echo 'Build failed'
      emailext subject: 'Build Failed',
           body: 'Build failed',
           to: 'team@company.com'
    }
  }
```

```
}
```

```
}
```

Pipeline Syntax

Common pipeline syntax and directives.

```
# Scripted Pipeline syntax
node {
    stage('Checkout') {
        checkout scm
    }

    stage('Build') {
        sh 'make build'
    }

    stage('Test') {
        sh 'make test'
        junit 'target/test-results/*.xml'
    }
}

# Parallel execution
stages {
    stage('Parallel Tests') {
        parallel {
            stage('Unit Tests') {
                steps {
                    sh 'make unit-test'
                }
            }
            stage('Integration Tests') {
                steps {
                    sh 'make integration-test'
                }
            }
        }
    }
}
```

Pipeline Triggers

Configure automatic pipeline triggers.

```
# Pipeline with triggers
pipeline {
    agent any

    triggers {
        // Poll SCM every 5 minutes
        pollSCM('H/5 * * * *')

        // Cron-like scheduling
        cron('H 2 * * *')  // Daily at 2 AM

        // Upstream job trigger
        upstream(upstreamProjects: 'upstream-job',
                threshold: hudson.model.Result.SUCCESS)
    }

    stages {
        stage('Build') {
            steps {
                sh 'make build'
            }
        }
    }
}
```

Plugin Management

Install and manage Jenkins plugins for extended functionality.

Plugin Installation: CLI

Install plugins using command line interface.

```
# Install plugin via CLI (requires restart)
java -jar jenkins-cli.jar -auth user:token install-plugin git

# Install multiple plugins
java -jar jenkins-cli.jar -auth user:token install-plugin \
  git maven-plugin docker-plugin

# Install from .hpi file
java -jar jenkins-cli.jar -auth user:token install-plugin \
  /path/to/plugin.hpi

# List installed plugins
java -jar jenkins-cli.jar -auth user:token list-plugins
```

Plugin Configuration

Configure plugins through web interface and global settings.

```
# Plugin installation via plugins.txt (for Docker)
# Create plugins.txt file:
git:latest
maven-plugin:latest
docker-plugin:latest
pipeline-stage-view:latest

# Use jenkins-plugin-cli tool
jenkins-plugin-cli --plugins git maven-plugin docker-plugin
```

Essential Plugins

Commonly used Jenkins plugins for different purposes.

```
# Build & SCM Plugins
git           # Git integration
github         # GitHub integration
maven-plugin      # Maven build support
gradle         # Gradle build support

# Pipeline Plugins
workflow-aggregator  # Pipeline plugin suite
pipeline-stage-view # Pipeline stage view
blue-ocean      # Modern UI for pipelines

# Deployment & Integration
docker-plugin    # Docker integration
kubernetes      # Kubernetes deployment
ansible       # Ansible automation

# Quality & Testing
junit         # JUnit test reports
jacoco       # Code coverage
sonarqube      # Code quality analysis
```

Plugin Management Web UI

Manage plugins through Jenkins web interface.

```
# Access Plugin Manager:
# 1. Navigate to Manage Jenkins
# 2. Click "Manage Plugins"
# 3. Use Available/Installed/Updates tabs
# 4. Search for plugins
# 5. Select and install
# 6. Restart Jenkins if required

# Plugin update process:
# 1. Check "Updates" tab
# 2. Select plugins to update
# 3. Click "Download now and install after restart"
```

User Management & Security

User Management

Create and manage Jenkins users.

```
# Enable Jenkins security:
# 1. Manage Jenkins → Configure Global Security
# 2. Enable "Jenkins' own user database"
# 3. Allow users to sign up (initial setup)
# 4. Set authorization strategy

# Create user via CLI (requires appropriate permissions)
# Users are typically created via web UI:
# 1. Manage Jenkins → Manage Users
# 2. Click "Create User"
# 3. Fill user details
# 4. Assign roles/permissions
```

Authentication & Authorization

Configure security realms and authorization strategies.

```
# Security configuration options:
# 1. Security Realm (how users authenticate):
#    - Jenkins' own user database
#    - LDAP
#    - Active Directory
#    - Matrix-based security
#    - Role-based authorization

# 2. Authorization Strategy:
#    - Anyone can do anything
#    - Legacy mode
#    - Logged-in users can do anything
#    - Matrix-based security
#    - Project-based Matrix Authorization
```

API Tokens

Generate and manage API tokens for CLI access.

```
# Generate API token:
# 1. Click username → Configure
# 2. API Token section
# 3. Click "Add new Token"
# 4. Enter token name
# 5. Generate and copy token

# Use API token with CLI
java -jar jenkins-cli.jar -auth username:api-token \
  -s http://localhost:8080 list-jobs

# Store credentials securely
echo "username:api-token" > ~/.jenkins-cli-auth
chmod 600 ~/.jenkins-cli-auth
```

Credentials Management

Manage stored credentials for jobs and pipelines.

```
# Manage credentials via CLI
java -jar jenkins-cli.jar -auth user:token \
  list-credentials system::system::jenkins

# Create credentials XML and import
java -jar jenkins-cli.jar -auth user:token \
  create-credentials-by-xml system::system::jenkins \
  < credential.xml

# Access credentials in pipelines
withCredentials([usernamePassword(
  credentialsId: 'my-credentials',
  usernameVariable: 'USERNAME',
  passwordVariable: 'PASSWORD'
)]) {
  sh 'docker login -u $USERNAME -p $PASSWORD'
}
```

Build Monitoring & Troubleshooting

Build Status & Logs

Monitor build status and access detailed logs.

```
# Check build status
java -jar jenkins-cli.jar -auth user:token console my-job

# Get build info
java -jar jenkins-cli.jar -auth user:token get-job my-job

# Monitor build queue
# Web UI: Jenkins Dashboard → Build Queue
# Shows pending builds and their status

# Build history access
# Web UI: Job → Build History
# Shows all previous builds with status
```

System Information

Get Jenkins system information and diagnostics.

```
# System information
java -jar jenkins-cli.jar -auth user:token version

# Node information
java -jar jenkins-cli.jar -auth user:token list-computers

# Groovy console (admin only)
# Manage Jenkins → Script Console
```

Log Analysis

Access and analyze Jenkins system logs.

```
# System logs location
# Linux: /var/log/jenkins/jenkins.log
# Windows: C:\Program Files\Jenkins\jenkins.out.log

# View logs
tail -f /var/log/jenkins/jenkins.log

# Log levels configuration
# Manage Jenkins → System Log
# Add new log recorder for specific components

# Common log locations:
sudo journalctl -u jenkins.service     # Systemd logs
sudo cat /var/lib/jenkins/jenkins.log  # Jenkins log file
```

Performance Monitoring

Monitor Jenkins performance and resource usage.

```
# Built-in monitoring
# Manage Jenkins → Load Statistics
# Shows executor utilization over time

# JVM monitoring
# Manage Jenkins → Manage Nodes → Master
# Shows memory, CPU usage, and system properties
```

```
# Execute Groovy scripts for system info:
println Jenkins.instance.version
println Jenkins.instance.getRootDir()
println System.getProperty("java.version")
```

```

# Build trends
# Install "Build History Metrics" plugin
# View build duration trends and success rates

# Disk usage monitoring
# Install "Disk Usage" plugin
# Monitor workspace and build artifact storage
```

Jenkins Configuration & Settings

Configure Jenkins daemon and system settings for optimal performance.

Global Configuration

Configure global Jenkins settings and tools.

```
# Global Tool Configuration
# Manage Jenkins → Global Tool Configuration
# Configure:
# - JDK installations
# - Git installations
# - Maven installations
# - Docker installations

# System Configuration
# Manage Jenkins → Configure System
# Set:
# - Jenkins URL
# - System message
# - # of executors
# - Quiet period
# - SCM polling limits
```

Environment Variables

Configure Jenkins environment variables and system properties.

```
# Built-in environment variables
BUILD_NUMBER        # Build number
BUILD_ID            # Build ID
JOB_NAME            # Job name
WORKSPACE           # Job workspace path
JENKINS_URL         # Jenkins URL
NODE_NAME           # Node name

# Custom environment variables
# Manage Jenkins → Configure System
# Global properties → Environment variables
# Add key-value pairs for global access
```

Jenkins Configuration as Code

Manage Jenkins configuration using JCasC plugin.

```
# JCasC configuration file (jenkins.yaml)
jenkins:
  systemMessage: "Jenkins configured as code"
  numExecutors: 4
  securityRealm:
    local:
      allowsSignup: false
      users:
        - id: "admin"
          password: "admin123"

# Apply configuration
# Set CASC_JENKINS_CONFIG environment variable
export CASC_JENKINS_CONFIG=/path/to/jenkins.yaml
```

Best Practices

Essential guidelines for efficient and secure Jenkins usage.

Security Best Practices

Keep your Jenkins instance secure and production-ready.

```
# Security recommendations:
# 1. Enable security and authentication
# 2. Use matrix-based authorization
# 3. Regular security updates
# 4. Limit user permissions
# 5. Use API tokens instead of passwords

# Secure Jenkins configuration:
# - Disable CLI over remoting
# - Use HTTPS with valid certificates
# - Regular backup of JENKINS_HOME
# - Monitor security advisories
# - Use credential plugins for secrets
```

Performance Optimization

Optimize Jenkins for better performance and scalability.

```
# Performance tips:
# 1. Use distributed builds with agents
# 2. Optimize build scripts and dependencies
# 3. Clean up old builds automatically
# 4. Use pipeline libraries for reusability
# 5. Monitor disk space and memory usage

# Build optimization:
# - Use incremental builds where possible
# - Parallel execution of stages
# - Artifact caching
# - Workspace cleanup
# - Resource allocation tuning
```

Reference: This cheatsheet covers essential Jenkins commands and modern practices for efficient continuous integration and deployment automation in development and production environments.