Jenkins Job Exploration — Step-by-Step (Freestyle → Pipeline)

> This guide walks you step-by-step from a simple Freestyle job to a full Declarative Pipe
> It is written to be concise, practical, and ready to use.

Prerequisites
1. Jenkins LTS installed and running.
2. Git plugin and Credentials plugin installed (usually included).
3. Admin access to Jenkins to create jobs and add credentials.
4. (Optional) Blue Ocean or Pipeline Stage View plugin for visual stage rendering.

---

1. Freestyle Job Basics

1.1 Create a Simple Echo Job
1. Jenkins → **New Item** → enter name `freestyle-echo` → choose **Freestyle project** –
2. In **Build** section → **Add build step** → **Execute shell**.
3. Paste:
```
echo "Hello from Jenkins!"
date
whoami
```
4. Save → **Build Now** → check **Console Output**.

---

2. Workspace File Operations

2.1 Create & View Files in Workspace
Add build step:
```
echo "creating testfile..."
echo "this file created at $(date)" > testfile.txt
cat testfile.txt
```
- After build, go to **Workspace** (if enabled) or view console to confirm.

---

3. Adding Git Repository

3.1 Configure Git SCM (Freestyle)
1. Edit the job → under **Source Code Management** select **Git**.
2. Repository URL (example public):
```
https://github.com/octocat/Hello-World.git
```
3. Branches to build: `*/main` or `*/master`.

3.2 Test Checkout
Add build step:
```
echo "Files checked out:"
ls -la
```

---

4. Credentials (for private repos)

4.1 Add Credentials
1. Manage Jenkins → **Credentials** → Global → **Add Credentials**.
2. Choose **Username with password** or **SSH Username with private key**.
3. Save, then select that credential in the job's SCM settings.

4.2 Alternative: Credentials Binding (for custom git commands)
1. Add **Secret text** credential (e.g., GIT_TOKEN).
2. In job: **Build Environment → Use secret text(s) or file(s)** → bind to env var `GIT_T
3. In shell step:
```
git clone https://$GIT_TOKEN@github.com/your-org/private-repo.git repo
cd repo
ls -la
```

---

5. Parameters (Make job reusable)

5.1 Add Build Parameters
1. In job config check **This project is parameterized**.
2. Add:
   - **String Parameter**: `BRANCH` (default `main`)
   - **Choice Parameter**: `ENV` (`dev`, `staging`, `prod`)
   - **Boolean Parameter**: `SKIP_TESTS` (default `false`)

5.2 Use Parameters in Build Step
```
echo "Building branch: $BRANCH"
echo "Target env: $ENV"
if [ "$SKIP_TESTS" = "false" ]; then
  echo "Running tests..."
else
  echo "Skipping tests"
fi
```

---

6. Build Triggers

6.1 Poll SCM
- Job → **Build Triggers** → **Poll SCM**.
- Schedule example (every 5 minutes):
```
H/5 * * * *
```

6.2 Build Periodically
- Job → **Build Triggers** → **Build periodically**.
- Example (daily around 2AM):
```
H 2 * * *
```

6.3 Webhook (Recommended)
- On GitHub/GitLab: create webhook to `http://<jenkins-host>/github-webhook/`.
- In job enable **GitHub hook trigger for GITScm polling**.
- Ensure Jenkins is reachable (use NAT/reverse-proxy or ngrok for local testing).

---

7. Simulated End-to-End Freestyle Pipeline
Add a shell build step to simulate stages:
```
echo "=== BUILD ==="
echo "Compiling..."
sleep 2

echo "=== TEST ==="
echo "Running tests..."
sleep 2

echo "=== PACKAGE ==="
echo "Packaging artifact..."
sleep 1

echo "=== DEPLOY ==="
echo "Deploying to $ENV..."
sleep 1
```
Post-build: Archive `target/*.txt` or any artifact you create.

---

8. Useful Freestyle Post-build Actions & Settings
- **Archive the artifacts** (e.g., `target/*.txt`).
- **Publish JUnit test results** (configure path like `target/test-results/*.xml`).
- **Discard old builds** to save disk space.
- **Workspace Cleanup Plugin** to clean before/after build.
- **Restrict where the project can run** (node labels).
- **Throttle builds** or disallow concurrent builds (job config).

---

9. Transition to Pipeline Job (Declarative)

9.1 Install Recommended Plugins
- Pipeline
- Pipeline Stage View (for stage visualization)
- Pipeline Graph View (optional)
- Credentials Binding (if using secrets)

9.2 Create a Pipeline Job
1. New Item → **Pipeline** → name `pipeline-demo` → OK.
2. In **Pipeline** section choose **Pipeline script** and paste the Declarative Jenkinsfil
below.

9.3 Minimal Declarative Jenkinsfile (copy-paste)
```groovy
pipeline {
  agent any

  parameters {
    string(name: 'BRANCH', defaultValue: 'main', description: 'Branch')
```

```
    choice(name: 'ENV', choices: ['dev','staging','prod'], description: 'Environment')
    booleanParam(name: 'SKIP_TESTS', defaultValue: false, description: 'Skip tests')
  }

  stages {
    stage('Checkout') {
      steps {
        echo "Checking out ${params.BRANCH}"
        git branch: params.BRANCH, url: 'https://github.com/your-username/sample-repo.git'
        sh 'ls -la'
      }
    }

    stage('Build') {
      steps {
        echo "Building..."
        sh 'echo "Compiling source..." && sleep 1'
      }
    }

    stage('Test') {
      when {
        expression { return !params.SKIP_TESTS }
      }
      steps {
        echo "Testing..."
        sh 'echo "Running tests" && sleep 1'
      }
    }

    stage('Package') {
      steps {
        echo "Packaging..."
        sh 'mkdir -p target && echo "artifact" > target/art.txt'
      }
    }

    stage('Deploy') {
      steps {
        echo "Deploying to ${params.ENV}"
        sh 'echo "Simulated deploy"'
      }
    }
  }

  post {
    always {
      junit 'target/test-results/*.xml'
      archiveArtifacts artifacts: 'target/*.txt', fingerprint: true
    }
  }
}
```

---

10. Enhancements & Best Practices

- **Use `withCredentials`** for sensitive data:
```groovy
```

```
withCredentials([usernamePassword(credentialsId: 'deploy-creds', usernameVariable: 'USER',
passwordVariable: 'PASS')]) {
  sh 'echo "Deploying with $USER"'
}
```
- **Prefer webhooks** over polling to reduce load.
- **Use shallow clone** in Git SCM for faster checkouts.
- **Run `mvn -B -DskipTests=true clean package`** in Build stage for Java projects.
- **Archive artifacts & publish test results** for traceability.
- **Restrict nodes via labels** for heavy workloads.
- **Fail fast**: use `set -e` in shell steps or `error` in Groovy when needed.

---

11. From Simulation to Real Spring Boot Build

Replace the Build and Test stages with:
```groovy
stage('Build') {
  steps {
    sh 'mvn -B -DskipTests=${params.SKIP_TESTS} clean package'
  }
}

stage('Test') {
  steps {
    junit 'target/surefire-reports/*.xml'
  }
}
```
And archive the generated jar:
```
archiveArtifacts artifacts: 'target/*.jar', fingerprint: true
```

---

12. Example: Full Pipeline for Spring Boot Hello World
1. Checkout repo (Jenkinsfile from SCM recommended).
2. Build with Maven.
3. Run unit tests and publish JUnit reports.
4. Archive `.jar`.
5. (Optional) Build Docker image and push to registry.
6. (Optional) Deploy to target environment (SSH, Kubernetes, or EC2).

---

13. Troubleshooting Tips
- **Stages not visible**: install *Pipeline Stage View* plugin and ensure pipeline uses De
syntax (`pipeline { stages { ... } }`).
- **Git credential errors**: verify credential type (token vs SSH key) and test clone from
Jenkins master/agent node.
- **Long-running steps**: configure proper timeouts or resource limits on agents.
- **Webhooks not firing**: ensure Jenkins is reachable from Git provider, or use a tunnel
Jenkins.

---

14. Summary & Next Steps
- Start with Freestyle to learn basic Jenkins features.

- Move to Declarative Pipeline for structured, versionable CI/CD.
- Convert simulated `echo` steps to real build/test/deploy commands for your Spring Boot a
- Use credentials store and webhooks for secure, efficient automation.

End of guide.