Jenkins CI/CD for Automation QA

Complete Interview Notes

Who this is for: Automation QA engineers preparing for an interview in the next 30-60 minutes.

What is inside: 18 chapters, 2 architecture diagrams, 3 complete pipelines, Maven and Git command tables, a troubleshooting matrix, 30 interview questions with answers, and a one-page cheat sheet.

How to use it: Read Chapters 1, 2, 5, 10 and 15 first - they cover 80% of what is asked. Then skim Chapter 17 and 18 right before the call.

Written from the perspective of a Senior DevOps Engineer and a Senior Automation QA Lead.

Contents

Every chapter ends with interview questions. Orange boxes are real-world examples. Grey boxes are copy-paste ready code.

1. CI, CD, and CI/CD Overview

Continuous Integration (CI) means every developer pushes code to a shared branch many times a day. Each push automatically triggers a build and a fast test run. The goal is simple: find broken code in minutes, not in weeks.

Continuous Delivery (CD) means every build that passes tests is packaged and is *ready* to release. A human clicks the button to release. **Continuous Deployment** removes even that click - if tests pass, the code goes to production automatically.

CI/CD pipeline = the automated chain: commit -> build -> test -> package -> deploy -> verify.

Term	What it automates	Who approves release	QA's job
CI	Build + unit + smoke tests on every commit	N/A (no release)	Keep smoke suite fast and stable
Continuous Delivery	Everything up to a deployable artifact	Human (release manager)	Own the regression gate before UAT
Continuous Deployment	Everything, including production deploy	Nobody - tests are the gate	Automated tests must be trusted 100%

Why an Automation QA must care

- Your test suite is the **quality gate**. If it is slow or flaky, the pipeline is ignored.
- Tests that only run on your laptop have no business value. Value starts when Jenkins runs them.
- Interviewers do not ask 'can you write a test?' They ask 'how does your test stop a bad build from reaching production?'

Real-world example: A 12-member team merged to *develop* only on Friday. Every Monday the nightly regression failed with 40+ errors and nobody knew which commit broke what. After moving to CI (build + 8-minute smoke suite on every push), failures were traced to a single commit within 10 minutes. Regression noise dropped by ~70%.

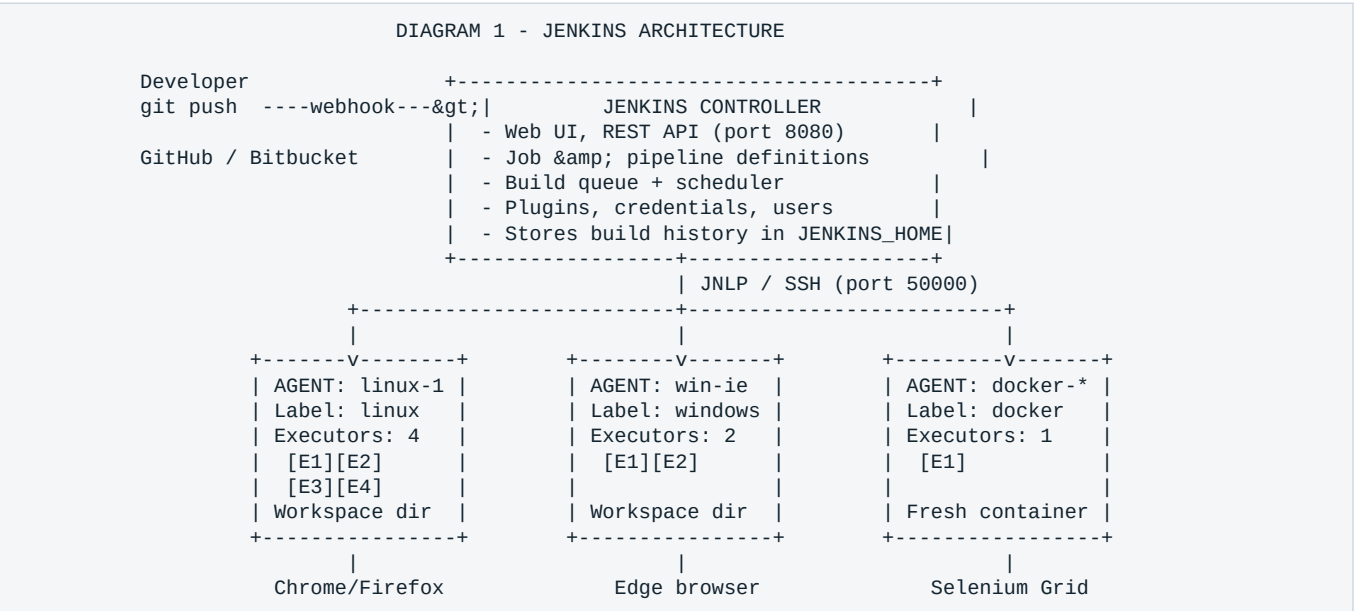
Interview Questions

Q1. What is the difference between Continuous Delivery and Continuous Deployment?
Q2. Your smoke suite takes 45 minutes. Why is that a CI problem, and how would you fix it?
Q3. Can you have CI without CD? Can you have CD without CI?
Q4. What test types belong in the CI stage vs the nightly stage?
Q5. How do you measure whether a CI/CD pipeline is healthy?

2. Jenkins Architecture (Controller, Agent, Node, Executor)

Jenkins uses a **controller-agent** model. The controller is the brain; agents are the hands. Modern Jenkins docs say **controller** instead of 'master' and **agent** instead of 'slave' - use the new words in an interview.

ASCII Diagram 1



Term	Meaning	Interview one-liner
Controller (Master)	Schedules jobs, serves UI, stores config and build history	Never run builds on it - keep it for orchestration only
Node	Any machine Jenkins can run work on (the controller itself is a node)	Node = the machine
Agent	A node that connects to the controller and executes builds	Agent = worker process on a node
Executor	One build slot on an agent. 4 executors = 4 parallel builds	Rule of thumb: executors = CPU cores
Label	A tag on an agent used by `agent { label 'linux' }`	How the pipeline picks the right machine
Workspace	Directory on the agent where the repo is checked out	Clean it with `cleanWs()` to avoid stale artifacts
JENKINS_HOME	Folder holding all jobs, config, plugins, build history	Back this up and you have backed up Jenkins

Real-world example: A QA team ran 30 Selenium tests directly on the controller. Chrome consumed all RAM, the Jenkins UI froze, and the build queue stalled for everyone. Fix: set controller executors to **0** and move all Selenium jobs to labelled agents.

Interview Questions

Q1. What is the difference between a node, an agent, and an executor?
Q2. Why should the number of executors on the controller be zero?
Q3. How does a pipeline choose which agent to run on?
Q4. What is stored inside JENKINS_HOME?
Q5. Two builds of the same job run at the same time - do they share a workspace?

3. Jenkins Installation (Brief)

Jenkins is a Java application. You need a supported JDK (Java 17 or 21 for current LTS lines). Three common ways to install it:

```
# 1) WAR file - fastest for a demo or local practice
java -jar jenkins.war --httpPort=8080

# 2) Debian / Ubuntu package - the usual production choice
sudo apt-get install -y openjdk-17-jdk
sudo apt-get install -y jenkins
sudo systemctl enable --now jenkins
sudo systemctl status jenkins

# 3) Docker - clean and repeatable
docker run -d --name jenkins -p 8080:8080 -p 50000:50000 \
  -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk17

# First-run unlock password
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

Item	Value / Path
Web UI port	8080 (change with `--httpPort`)
Agent (inbound JNLP) port	50000
JENKINS_HOME (Linux pkg)	/var/lib/jenkins
JENKINS_HOME (Docker)	/var/jenkins_home
Plugins a QA needs on day 1	Git, Pipeline, Maven Integration, HTML Publisher, Allure, Email Extension, Credentials Binding, Blue Ocean (optional)
Global tools to configure	Manage Jenkins > Tools > JDK, Maven, Git (names used in `tools { }` block)

Interview Questions

Q1. Which port do inbound agents use to connect to the controller?
Q2. Where is the initial admin password stored?
Q3. How do you upgrade Jenkins safely without losing jobs?
Q4. Name three plugins an automation QA cannot work without.

4. Freestyle Job vs Pipeline Job

Aspect	Freestyle Job	Pipeline Job
Definition	Configured by clicking in the Jenkins UI	Written as code in a `Jenkinsfile`
Version control	Config lives in Jenkins XML, not in your repo	Jenkinsfile lives with the test code, reviewed in PRs
Stages	One long list of build steps, no stage view	Named stages, visual stage-by-stage status
Parallelism	Very limited	Native `parallel { }` block
Restart / resume	No	Yes - restart from a failed stage
Complex logic	Needs shell hacks	Groovy: `when`, `try/catch`, loops, functions
Best use	Quick one-off task, legacy jobs	Everything real: build, test, deploy, multi-env

Answer to give: 'Freestyle is UI-configured and fine for a single task. Pipeline is Pipeline-as-Code - the Jenkinsfile sits in Git next to my tests, so the pipeline is reviewed, versioned and rolled back like any other code.'

Real-world example: A team had 40 freestyle jobs. Migrating one job to prod meant re-clicking 30 fields and someone always forgot a checkbox. After moving to a shared Jenkinsfile with an `ENV` parameter, 40 jobs became 1 pipeline + 5 parameter sets.

Interview Questions

Q1. Why is Pipeline-as-Code better than a freestyle job?
Q2. Can a freestyle job run stages in parallel?
Q3. What is a Multibranch Pipeline and when do you use it?
Q4. How do you restart a failed pipeline from a specific stage?

5. Jenkinsfile (Declarative Pipeline)

Declarative pipeline has a fixed structure and is easier to read. Scripted pipeline is pure Groovy and gives more freedom. **Use declarative unless you have a reason not to.**

Block	Purpose
`pipeline { }`	Root block. Everything lives inside it
`agent`	Where the pipeline runs: `any`, `none`, `label 'linux'`, `docker { image '...' }`
`tools`	Auto-install JDK / Maven configured in Global Tools
`parameters`	User inputs shown on 'Build with Parameters'
`environment`	Key-value env vars, including credentials
`options`	Timeouts, retries, build rotation, timestamps
`triggers`	cron, pollSCM, upstream
`stages / stage / steps`	The actual work
`when`	Conditional stage execution
`post`	`always / success / failure / unstable / changed` - reports and email go here

Sample Jenkinsfile (complete, Selenium + TestNG + Maven)

Sample Jenkinsfile

```
pipeline {
  agent { label 'linux' }

  tools { jdk 'jdk17'; maven 'maven3' }

  parameters {
    choice(name: 'ENVIRONMENT', choices: ['dev','qa','uat','preprod'], description: 'Target env')
    choice(name: 'BROWSER',     choices: ['chrome','firefox','edge'],  description: 'Browser')
    choice(name: 'SUITE',       choices: ['smoke','regression','sanity'], description: 'TestNG xml')
    booleanParam(name: 'HEADLESS', defaultValue: true, description: 'Run headless')
  }

  environment {
    BASE_URL   = "https://${params.ENVIRONMENT}.shop-demo.com"
    APP_CREDS  = credentials('qa-app-login')     // -&gt; APP_CREDS_USR / APP_CREDS_PSW
    MAVEN_OPTS = '-Xmx1024m'
  }

  options {
    timeout(time: 60, unit: 'MINUTES')
    buildDiscarder(logRotator(numToKeepStr: '20'))
    timestamps()
    disableConcurrentBuilds()
  }

  triggers { cron('H 1 * * 1-5') }              // nightly, Mon-Fri, ~01:00

  stages {
    stage('Checkout') {
      steps {
        cleanWs()
        git branch: 'main', url: 'https://github.com/acme/shop-automation.git',
            credentialsId: 'github-pat'
      }
    }

    stage('Build') {
      steps { sh 'mvn -B clean compile' }
    }

    stage('Test') {
      steps {
        sh """
          mvn -B test \
            -DsuiteXmlFile=src/test/resources/suites/${params.SUITE}.xml \
            -Dbrowser=${params.BROWSER} \
            -Dheadless=${params.HEADLESS} \
            -Dbase.url=${BASE_URL} \
            -Dapp.user=${APP_CREDS_USR} -Dapp.pass=${APP_CREDS_PSW}
        """
      }
    }

    stage('Deploy to QA') {
      when { expression { params.ENVIRONMENT == 'qa' &amp;&amp; currentBuild.result == null } }
      steps { sh './scripts/deploy.sh qa' }
    }
  }

  post {
    always {
      junit testResults: '**/target/surefire-reports/*.xml', allowEmptyResults: true
      allure includeProperties: false, results: [[path: 'target/allure-results']]
      archiveArtifacts artifacts: 'target/screenshots/**', allowEmptyArchive: true
    }
    failure {
      emailext to: 'qa-team@acme.com',
               subject: "FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
               body: "Suite ${params.SUITE} failed. Report: ${env.BUILD_URL}allure/"
    }
    cleanup { cleanWs() }
  }
}
```

Note the credential rule: `credentials('id')` on a username-password credential creates two masked variables, `_USR` and `_PSW`. Never print them with `echo`.

Interview Questions

Q1. Difference between declarative and scripted pipeline?
Q2. What runs inside the `post { always }` block and why?
Q3. How do you inject a password into a Maven command without exposing it in the console log?
Q4. What does `agent none` mean at pipeline level?
Q5. How do you make a stage run only on the main branch?

6. Git + Maven Integration

Jenkins pulls the test code from Git, then Maven compiles it, resolves dependencies and runs TestNG through the **Surefire** plugin.

Important Git commands

Command	What it does
`git clone <url>`	Copy the remote repo locally
`git checkout -b feature/login`	Create and switch to a branch
`git status` / `git diff`	See changed files / actual changes
`git add . && git commit -m "msg"`	Stage and commit
`git pull --rebase origin main`	Get latest main without a merge commit
`git push origin feature/login`	Push branch to remote
`git merge main` / `git rebase main`	Bring main into your branch
`git log --oneline -10`	Last 10 commits, short form
`git revert <sha>`	Undo a commit safely (creates a new commit)
`git reset --hard origin/main`	Throw away local changes (dangerous)
`git stash` / `git stash pop`	Park uncommitted work
`git tag -a v1.4.0 -m "rel"`	Tag a release commit

Important Maven commands

Command	What it does
`mvn clean`	Delete the `target/` folder
`mvn compile`	Compile `src/main/java`
`mvn test-compile`	Compile test sources
`mvn test`	Run tests via Surefire (TestNG/JUnit)
`mvn package`	Build the jar/war (runs tests first)
`mvn install`	Copy artifact into the local ~/.m2 repo
`mvn clean test -DsuiteXmlFile=smoke.xml`	Run one TestNG suite
`mvn test -Dgroups=smoke`	Run tests by TestNG group
`mvn test -Dtest=LoginTest#validLogin`	Run a single test method
`mvn -B -U clean verify`	Batch mode, force-update snapshots, run integration tests
`mvn dependency:tree`	Debug jar version conflicts
`mvn test -DskipTests=false -Dmaven.test.failure.ignore=true`	Do not fail the build on test failure (report still published)

Key Maven lifecycle order: validate -> compile -> test -> package -> verify -> install -> deploy. Each phase runs all phases before it.

Real-world example: A pipeline used `mvn test` and the build showed SUCCESS even though 6 tests failed - because `maven.test.failure.ignore` was set to true in the POM. Nobody looked at the report for two sprints. Fix: keep the flag false, and let `junit` step mark the build UNSTABLE.

Interview Questions

Q1. What is the difference between `mvn package` and `mvn install`?
Q2. Which Maven plugin executes your TestNG suite?
Q3. How do you run only smoke tests from the command line?
Q4. Why use `-B` (batch mode) in Jenkins?
Q5. Your build fails with a NoSuchMethodError at runtime. Which Maven command helps first?

7. Build Triggers (Manual, Webhook, Schedule)

Trigger	How it works	When to use
Manual	Click 'Build' / 'Build with Parameters'	Ad-hoc runs, release deploys, debugging
SCM Webhook (push)	Git server sends an HTTP POST to `/github-webhook/`; Jenkins builds immediately	Best for CI. Instant, no polling load
Poll SCM	`pollSCM('H/5 * * * *')` - Jenkins asks Git every 5 min 'anything new?'	Fallback when the Git server cannot reach Jenkins
Schedule (cron)	`cron('H 1 * * 1-5')` - run at a fixed time	Nightly regression, weekly cross-browser
Upstream job	`build job: 'qa-regression'` or `upstream()` trigger	Deploy job finishes -> test job starts
Remote / API	`curl -X POST .../job/x/build?token=T`	Triggered by an external tool

Jenkins cron syntax

```
MINUTE  HOUR  DAY-OF-MONTH  MONTH  DAY-OF-WEEK      (0=Sunday .. 7=Sunday)

H 1 * * 1-5      -&gt; once between 01:00 and 01:59, Monday to Friday
H/15 * * * *     -&gt; every 15 minutes
H 22 * * *       -&gt; every night around 22:00
H 2 * * 6        -&gt; Saturday early morning (weekly cross-browser suite)
@midnight        -&gt; alias, sometime between 00:00 and 02:59

'H' = hash. Jenkins spreads load using the job name, so 50 jobs do not all start at 01:00.
Always prefer H over a fixed number.
```

Real-world example: A team scheduled 12 regression jobs at exactly `0 1 * * *`. All 12 launched at 01:00, the agent ran out of memory and 9 jobs failed with Chrome crashes. Replacing `0` with `H` spread them across the hour and the failures disappeared.

Interview Questions

Q1. Webhook vs Poll SCM - which is better and why?
Q2. What does `H` mean in Jenkins cron?
Q3. How do you trigger a downstream test job after a deploy job succeeds?
Q4. How would you trigger a Jenkins build from a shell script?

8. Environment Management (DEV, QA, UAT, PRE-PROD, PROD)

Env	Purpose	Who uses it	Tests that run	Data
DEV	Developer integration	Developers	Unit + smoke	Fake / seeded
QA	Functional verification	QA team	Full regression, API, UI	Controlled test data
UAT	Business sign-off	Business users, BA	Business scenarios, exploratory	Prod-like masked data
PRE-PROD (Staging)	Last technical rehearsal	QA + DevOps	Smoke, performance, security	Prod-like, same config
PROD	Live customers	Everyone	Smoke / health check only	Real data - never write test data

How Jenkins handles environments

- One Jenkinsfile, one `ENVIRONMENT` parameter - never one job per env.

- Keep per-env values in property files: `config/qa.properties`, `config/uat.properties`, and pass `-Denv=qa`.
- Store per-env passwords in Jenkins **Credentials**, scoped by folder, never in Git.
- Use `when { environment name: 'ENVIRONMENT', value: 'preprod' }` to gate risky stages.
- Add a manual gate before PROD: `input message: 'Deploy to PROD?', submitter: 'release-managers'`.

```
environment {
  BASE_URL = "https://${params.ENVIRONMENT}.shop-demo.com"
  DB_CREDS = credentials("db-${params.ENVIRONMENT}")   // db-qa, db-uat, db-preprod
}
```

Real-world example: Someone ran the regression suite against PROD by picking the wrong dropdown value; it created 300 dummy orders. Fix applied: PROD removed from the choice list of the regression job, and a separate 'prod-smoke' pipeline with read-only tests and an `input` approval step was created.

Interview Questions

Q1. How do you run the same suite against QA and UAT without duplicating the job?
Q2. Where do you store environment-specific passwords?
Q3. Why should full regression never run on PROD?
Q4. What is a manual approval gate in a declarative pipeline?
Q5. How do you prevent test data leaking into production?

9. Parameterized Builds (Browser, Environment, Suite)

A parameterized build lets one pipeline serve many combinations. The job page shows **Build with Parameters** instead of **Build Now**.

Parameter type	Syntax	Typical QA use
choice	`choice(name:'BROWSER', choices:['chrome','firefox'])`	Browser, environment, suite
string	`string(name:'TAG', defaultValue:'smoke')`	TestNG group, release tag
booleanParam	`booleanParam(name:'HEADLESS', defaultValue:true)`	Headless on/off, skip cleanup
password	`password(name:'API_KEY')`	One-off secret (prefer Credentials)
text	`text(name:'NOTES')`	Release notes
Active Choices (plugin)	Groovy script generates the list	Suite list read from the repo

```
parameters {
  choice(name: 'ENVIRONMENT', choices: ['dev','qa','uat','preprod'], description: 'Target env')
  choice(name: 'BROWSER',     choices: ['chrome','firefox','edge'],  description: 'Browser')
  choice(name: 'SUITE',       choices: ['smoke','sanity','regression'], description: 'Suite')
  booleanParam(name: 'HEADLESS', defaultValue: true)
}

stage('Test') {
  steps {
    sh "mvn -B clean test -DsuiteXmlFile=suites/${params.SUITE}.xml " +
       "-Dbrowser=${params.BROWSER} -Dheadless=${params.HEADLESS} " +
       "-Denv=${params.ENVIRONMENT}"
  }
}
```

On the test-code side, read the values in your `BaseTest`:

```
String browser  = System.getProperty("browser", "chrome");
boolean headless = Boolean.parseBoolean(System.getProperty("headless", "true"));

ChromeOptions options = new ChromeOptions();
if (headless) options.addArguments("--headless=new", "--window-size=1920,1080",
                                   "--no-sandbox", "--disable-dev-shm-usage");
WebDriver driver = new ChromeDriver(options);
```

Real-world example: Before parameters: 9 jobs (3 browsers x 3 envs). After: 1 pipeline. Maintenance dropped from 9 Jenkinsfiles to 1, and a new browser was added by editing one line.

Interview Questions

Q1. How do you pass a Jenkins parameter into a TestNG test?
Q2. Difference between a `password` parameter and a Jenkins credential?
Q3. How can you make a parameter list load dynamically from the repo?
Q4. What is `params.X` versus `env.X`?

10. Running Selenium / TestNG Automation from Jenkins

The four things that must be true

- **Browser + driver exist on the agent** (or you use a Selenium Grid / Docker container).
- **No display server needed** - run headless, or wrap with `xvfb`.
- **Suite selection is dynamic** - `-DsuiteXmlFile=...`.
- **Results are always published**, even when tests fail (that is what `post { always }` is for).

pom.xml - Surefire wiring

```
&lt;plugin&gt;
  &lt;groupId&gt;org.apache.maven.plugins&lt;/groupId&gt;
  &lt;artifactId&gt;maven-surefire-plugin&lt;/artifactId&gt;
  &lt;version&gt;3.2.5&lt;/version&gt;
  &lt;configuration&gt;
    &lt;suiteXmlFiles&gt;
      &lt;suiteXmlFile&gt;${suiteXmlFile}&lt;/suiteXmlFile&gt;   &lt;!-- passed from Jenkins --&gt;
    &lt;/suiteXmlFiles&gt;
    &lt;systemPropertyVariables&gt;
      &lt;browser&gt;${browser}&lt;/browser&gt;
      &lt;env&gt;${env}&lt;/env&gt;
    &lt;/systemPropertyVariables&gt;
  &lt;/configuration&gt;
&lt;/plugin&gt;
```

testng.xml - suite file

```
&lt;suite name="regression" parallel="classes" thread-count="4"&gt;
  &lt;listeners&gt;
    &lt;listener class-name="io.qameta.allure.testng.AllureTestNg"/&gt;
  &lt;/listeners&gt;
  &lt;test name="checkout-flow"&gt;
    &lt;classes&gt;
      &lt;class name="com.acme.tests.LoginTest"/&gt;
      &lt;class name="com.acme.tests.CheckoutTest"/&gt;
    &lt;/classes&gt;
  &lt;/test&gt;
&lt;/suite&gt;
```

Running against a Grid or Docker

```
// Option A: point tests at a remote grid
sh "mvn -B test -Dremote=true -Dgrid.url=http://selenium-hub:4444/wd/hub -Dbrowser=chrome"

// Option B: give the pipeline its own browser container
agent { docker { image 'maven:3.9-eclipse-temurin-17'; args '--shm-size=2g' } }
```

Always add `--shm-size=2g` for Chrome in Docker. The default 64 MB shared memory is the number one cause of `session not created / tab crashed` errors in CI.

Real-world example: Tests passed locally but every Jenkins run failed with `unknown error: DevToolsActivePort file doesn't exist`. Root cause: Chrome started in a container without `--no-sandbox` and `--disable-dev-shm-usage`. Adding those two flags fixed 100% of the failures.

Interview Questions

Q1. Why do Selenium tests fail on a Jenkins agent but pass on your laptop?
Q2. What is headless mode and what is one risk of using it?
Q3. How do you make Jenkins mark a build UNSTABLE (yellow) instead of FAILED when tests fail?
Q4. How do you attach a screenshot of a failed test to the Jenkins build?
Q5. How would you run tests on a browser Jenkins agents do not have installed?

11. Allure / HTML Report Publishing

Report	Plugin / step	Gives you
JUnit / TestNG XML	`junit '**/target/surefire-reports/*.xml'`	Pass/fail trend graph, sets build UNSTABLE on failures
Allure	`allure results:[[path:'target/allure-results']]`	Rich report: steps, screenshots, history, flaky detection
HTML Publisher	`publishHTML(...)`	Any static HTML (ExtentReports, Cucumber, custom)
Artifacts	`archiveArtifacts 'target/screenshots/**'`	Download screenshots, logs, videos

```
post {
  always {
    junit testResults: '**/target/surefire-reports/*.xml', allowEmptyResults: true

    allure includeProperties: false,
           jdk: '',
           results: [[path: 'target/allure-results']]

    publishHTML(target: [
      reportDir  : 'target/site/extent',
      reportFiles: 'index.html',
      reportName : 'Extent Report',
      keepAll    : true,
      allowMissing: false,
      alwaysLinkToLastBuild: true
    ])

    archiveArtifacts artifacts: 'target/screenshots/**/*.png', allowEmptyArchive: true
  }
}
```

Allure setup checklist

- Add `allure-testng` dependency + `aspectjweaver` argLine in the POM.
- Install the **Allure Jenkins Plugin** and configure the Allure commandline tool (Manage Jenkins > Tools).
- Attach screenshots on failure with `Allure.addAttachment("screenshot", new ByteArrayInputStream(bytes))`.
- `allure-results` is raw JSON; the plugin turns it into HTML and keeps **history** across builds so you can see flaky tests.
- If the HTML report shows no CSS, it is Jenkins **Content Security Policy**. Fix it in script console or use the Allure plugin instead of publishHTML.

Real-world example: Reports were published only in `post { success }`. When a regression failed, the team had no report to debug with - only console logs. Moving `junit` and `allure` to `post { always }` is the single highest-value fix.

Interview Questions

Q1. Why publish reports in `always` and not `success`?
Q2. Difference between the JUnit step and the Allure report?
Q3. Allure report opens but is blank - what do you check?
Q4. How does Allure show test history and flakiness?

12. Email Notifications

Item	Detail
Plugins	**Mailer** (basic `mail` step) and **Email Extension** (`emailext`, templates, attachments)
Configure	Manage Jenkins > System > Extended E-mail Notification (SMTP host, port 587, TLS, credentials)
Where to call	`post { failure } / post { unstable } / post { changed }`
Anti-noise rule	Do not mail on every success. Mail on **failure** and on **fixed** (`changed`)

```
        post {
          unstable {
            emailext to: 'qa-team@acme.com',
              subject: "UNSTABLE: ${env.JOB_NAME} #${env.BUILD_NUMBER} (${params.ENVIRONMENT})",
              mimeType: 'text/html',
              body: """&lt;h3&gt;Some tests failed&lt;/h3&gt;
                       &lt;p&gt;Suite: ${params.SUITE} | Browser: ${params.BROWSER}&lt;/p&gt;
                       &lt;p&gt;&lt;a href="${env.BUILD_URL}allure/"&gt;Open Allure report&lt;/a&gt;&lt;/p&gt;""",
              attachLog: true,
              compressLog: true,
              attachmentsPattern: 'target/screenshots/**/*.png'
          }
          failure {
            emailext to: 'qa-team@acme.com, devops@acme.com',
              recipientProviders: [culprits(), developers(), requestor()],
              subject: "FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
              body: "Build failed before tests ran. Console: ${env.BUILD_URL}console"
          }
          fixed { emailext to: 'qa-team@acme.com', subject: "FIXED: ${env.JOB_NAME}", body: 'Back to green.' }
        }
```

`culprits()` mails whoever committed since the last good build - the fastest way to get the right person to look.

> **Real-world example:** A nightly job emailed 30 people on every run, pass or fail. Within a month everyone had an inbox rule to delete it, and a real 2-day outage went unnoticed. Rule: **alert only on state change**.

Interview Questions

Q1. Difference between the `mail` step and `emailext`?
Q2. How do you email only the developers who broke the build?
Q3. When should a build be UNSTABLE rather than FAILURE?
Q4. How do you attach the Allure link and screenshots to the failure mail?

13. Parallel Execution (Basic)

There are **two different parallelisms**, and mixing them up is a classic interview trap.

Level	Controlled by	Runs on	Use for
Jenkins `parallel`	Jenkinsfile `parallel { }`	Different agents / executors	Cross-browser, cross-env, split suites
TestNG parallel	`parallel="methods|classes|tests"` + `thread-count`	Threads inside one JVM on one agent	Speeding up one suite

```
        stage('Cross-browser regression') {
          parallel {
            stage('Chrome') {
              agent { label 'linux' }
              steps { sh 'mvn -B clean test -Dbrowser=chrome -DsuiteXmlFile=suites/regression.xml' }
              post { always { junit '**/surefire-reports/*.xml' } }
            }
            stage('Firefox') {
              agent { label 'linux' }
              steps { sh 'mvn -B clean test -Dbrowser=firefox -DsuiteXmlFile=suites/regression.xml' }
              post { always { junit '**/surefire-reports/*.xml' } }
            }
            stage('Edge') {
              agent { label 'windows' }
              steps { bat 'mvn -B clean test -Dbrowser=edge -DsuiteXmlFile=suites/regression.xml' }
            }
          }
        }
```

Rules for safe parallel tests

- **No shared static WebDriver.** Use `ThreadLocal<WebDriver>`.
- **No shared test data.** Two threads logging in as the same user will fight.
- Each parallel branch needs its **own workspace** - Jenkins gives that automatically per agent.
- `failFast true` inside `parallel` stops the rest when one branch fails - good for smoke, bad for nightly regression.
- Do not set `thread-count` higher than the agent's cores; Chrome is RAM-hungry (~500 MB per instance).

Real-world example: A 3-hour regression suite was split into 4 parallel Jenkins branches by module (login, catalog, cart, payments) across 2 agents. Wall time dropped to 52 minutes. The only code change needed was replacing a static driver with ThreadLocal.

Interview Questions

Q1. Difference between Jenkins `parallel` and TestNG `parallel`?
Q2. Why is a static WebDriver dangerous in parallel runs?
Q3. What does `failFast true` do?
Q4. You parallelised the suite and flakiness went up. What are the likely causes?

14. Common Pipeline Failures and Troubleshooting

Symptom	Likely cause	Fix
`DevToolsActivePort file doesn't exist`	Chrome in container without sandbox / shm	Add `--no-sandbox --disable-dev-shm-usage`, `--shm-size=2g`
`SessionNotCreatedException: version mismatch`	chromedriver older than Chrome	Use WebDriverManager / Selenium Manager, or pin browser version in Docker
Build hangs forever	No `timeout`; browser waiting for a modal	`options { timeout(time:60, unit:'MINUTES') }`
`No such file: target/allure-results`	Tests never ran, or wrong workspace path	`allowEmptyResults`, check pwd, check the build stage failed earlier
Build is GREEN but tests failed	`maven.test.failure.ignore=true` and no `junit` step	Add `junit` step so failures mark it UNSTABLE
`Permission denied: ./deploy.sh`	File not executable after checkout	`sh 'chmod +x deploy.sh'` or `sh 'bash deploy.sh'`
Stale test results / old jars	Dirty workspace reused	`cleanWs()` and `mvn clean`
`No space left on device`	Old builds, screenshots, Docker images	`buildDiscarder(logRotator(numToKeepStr: '20'))`, prune Docker
Credentials printed in log	`echo` of a secret, or secret in a `sh` string with `set -x`	Use `withCredentials`; never echo; add `set +x`
Webhook does not trigger	Jenkins URL not reachable / wrong endpoint / token	Check GitHub webhook 'Recent Deliveries' for the HTTP response code
`Agent went offline` mid-build	Agent OOM (Chrome), or network drop	Reduce executors/thread-count, add swap, monitor RAM

Symptom	Likely cause	Fix
Flaky test only in Jenkins	Timing, smaller screen, no cookies, different timezone/locale	Explicit waits, fix window size `1920x1080`, set TZ/locale

How to debug, in order

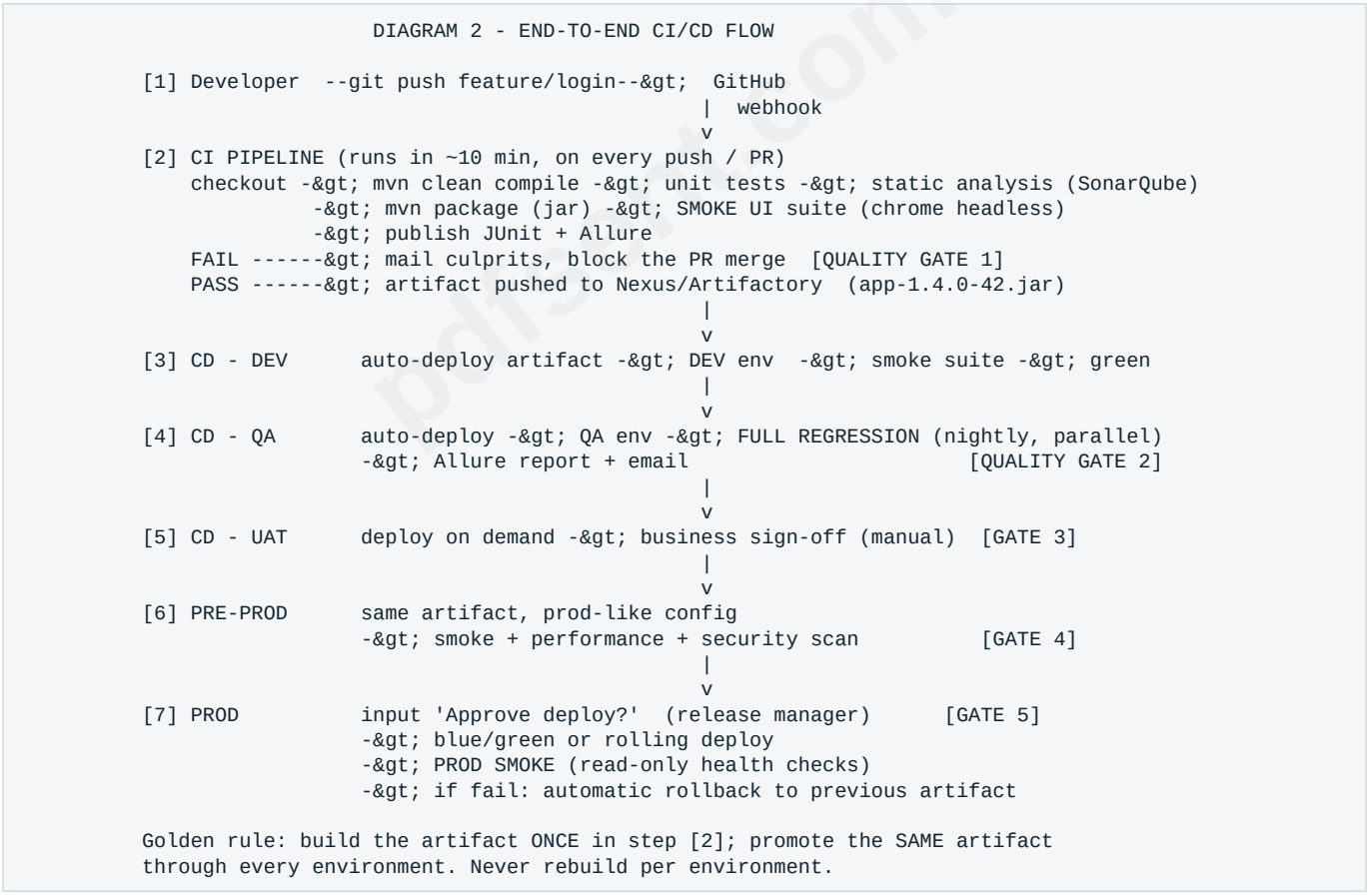
- Read the **first** error in the console log, not the last.
- Reproduce the exact `mvn` command from the log on the agent by hand.
- Compare agent vs laptop: Java version, browser version, env vars (`sh 'printenv | sort'`).
- Check the workspace: `sh 'ls -R target | head -50'`.
- Re-run with `-X` (Maven debug) only after the cheap checks.

Interview Questions

Q1. A test passes locally and fails in Jenkins. Walk me through your debugging.
Q2. How do you stop a stuck build automatically?
Q3. The disk on the controller is full. What do you clean first?
Q4. How do you prevent a secret from appearing in the console output?
Q5. What is the difference between FAILURE, UNSTABLE, and ABORTED?

15. Real-Time CI/CD Flow from Developer to Production

ASCII Diagram 2



Real-world pipeline example 1 - PR validation (fast gate, ~8 minutes)

```
pipeline {
  agent { label 'linux' }
  tools { jdk 'jdk17'; maven 'maven3' }
  options { timeout(time: 20, unit: 'MINUTES'); timestamps() }

  stages {
    stage('Checkout') { steps { cleanWs(); checkout scm } }
    stage('Build')    { steps { sh 'mvn -B clean compile' } }
    stage('Unit')     { steps { sh 'mvn -B test -Dgroups=unit' } }
    stage('Smoke UI') {
      steps {
        sh 'mvn -B test -DsuiteXmlFile=suites/smoke.xml -Dbrowser=chrome -Dheadless=true'
      }
    }
  }
  post {
    always  { junit '**/target/surefire-reports/*.xml' }
    failure { emailext recipientProviders: [culprits()],
                       subject: "PR build failed: ${env.BRANCH_NAME}",
                       body: "${env.BUILD_URL}" }
  }
}
```

Real-world pipeline example 2 - Nightly cross-browser regression

```
pipeline {
  agent none
  parameters { choice(name:'ENVIRONMENT', choices:['qa','uat']) }
  triggers   { cron('H 1 * * 1-5') }
  options    { buildDiscarder(logRotator(numToKeepStr: '30')) }

  stages {
    stage('Regression') {
      failFast false
      parallel {
        stage('Chrome')  { agent { label 'linux' }
          steps { sh "mvn -B clean test -Dbrowser=chrome  -Denv=${params.ENVIRONMENT} \
                     -DsuiteXmlFile=suites/regression.xml" } }
        stage('Firefox') { agent { label 'linux' }
          steps { sh "mvn -B clean test -Dbrowser=firefox -Denv=${params.ENVIRONMENT} \
                     -DsuiteXmlFile=suites/regression.xml" } }
        stage('Edge')    { agent { label 'windows' }
          steps { bat "mvn -B clean test -Dbrowser=edge -Denv=%ENVIRONMENT% ^
                      -DsuiteXmlFile=suites/regression.xml" } }
      }
    }
  }
  post {
    always   { junit '**/target/surefire-reports/*.xml'
               allure results: [[path: 'target/allure-results']] }
    unstable { emailext to: 'qa-team@acme.com', attachLog: true,
                        subject: "Nightly regression UNSTABLE #${env.BUILD_NUMBER}",
                        body: "Report: ${env.BUILD_URL}allure/" }
    fixed    { emailext to: 'qa-team@acme.com', subject: 'Nightly regression back to GREEN',
                        body: 'No action needed.' }
  }
}
```

Interview Questions

Q1. Why must the same artifact be promoted across environments instead of rebuilding?
Q2. Where exactly does QA automation sit in this flow?
Q3. What happens if the PROD smoke test fails after deploy?
Q4. Which stages should block a PR merge, and which should not?
Q5. How long should the CI gate take, and why does that number matter?

16. Jenkins Best Practices

Pipeline

- Keep the **Jenkinsfile in the repo** with the tests. Review it in PRs.
- Controller executors = **0**. Builds run only on agents.

- Always set `timeout` and `buildDiscarder`.
- `cleanWs()` at start or in `post { cleanup }`.
- Put reusable logic in a **Shared Library** (`vars/qaPipeline.groovy`), not copy-pasted across 20 Jenkinsfiles.
- Use `-B` (batch mode) for Maven so logs stay readable.

Security

- Secrets only in **Credentials** + `withCredentials`. Never in Git, never in `echo`.
- Role-Based Authorization: QA can build, only release managers can approve PROD.
- Keep Jenkins and plugins patched; disable unused plugins.
- Do not run agents as root.

Test automation specific

- **Fast gate, slow gate.** Smoke (<10 min) on every commit; regression nightly.
- **Zero tolerance for flaky tests.** Quarantine them into a separate suite the same day; a flaky suite destroys trust in the pipeline faster than a bug does.
- Never use `Thread.sleep()`. Explicit waits only.
- One test = one purpose = independent test data. No test order dependency.
- Publish reports in `post { always }`.
- Fail the build on test failure. A green build with failing tests is worse than no pipeline.

Maintenance

Practice	Why
Back up JENKINS_HOME	It is the whole Jenkins state
Pin plugin versions / test upgrades in a staging Jenkins	Plugin upgrades break pipelines
Monitor agent disk and RAM	Chrome fills both silently
Archive only what you need	Screenshots and videos eat disk fast
Track pipeline metrics: duration, failure rate, flaky rate	You cannot improve what you do not measure

Interview Questions

Q1. What is a Jenkins Shared Library and why use one?
Q2. How do you keep secrets out of the build log?
Q3. How do you handle a flaky test that fails 1 in 10 runs?
Q4. Why should the controller not run builds?
Q5. What would you back up to restore Jenkins after a disk failure?

17. Top 30 Jenkins Interview Questions with Answers

Answer in two sentences: **what it is**, then **why it matters in a real pipeline**. Long answers lose the room.

Q1. What is Jenkins?
An open-source automation server written in Java. It orchestrates build, test and deploy steps, triggered by events like a Git push or a schedule.

Q2. Why is Jenkins still used when GitLab CI / GitHub Actions exist?
Huge plugin ecosystem (1800+), runs on-premise behind a firewall, and can drive any tool. Trade-off: you maintain the server yourself.

Q3. Explain Jenkins architecture.
Controller schedules and stores config; agents execute builds. Each agent has executors (build slots) and a workspace. Agents are selected by label.

Q4. Controller vs agent vs node vs executor?
Node = a machine. Agent = the worker on a node. Executor = one build slot on an agent. Controller = the orchestrating node.

Q5. What is JENKINS_HOME?
The directory holding jobs, plugins, credentials, config and build history. Backing it up backs up Jenkins.

Q6. Freestyle vs Pipeline job?
Freestyle is UI-configured, no stages, no version control. Pipeline is code in a Jenkinsfile stored in Git, supports stages, parallel, restart and code review.

Q7. Declarative vs scripted pipeline?
Declarative has a fixed `pipeline { }` structure, built-in validation and `post` conditions. Scripted is raw Groovy - more power, less structure. Prefer declarative.

Q8. What is a Jenkinsfile?
A text file in the repo root that defines the pipeline as code. It is versioned and reviewed with the application/test code.

Q9. What is a Multibranch Pipeline?
Jenkins scans a repo, and for every branch or PR containing a Jenkinsfile it auto-creates a job. Perfect for PR validation.

Q10. Mandatory sections of a declarative pipeline?
`pipeline`, `agent`, `stages`, and at least one `stage` with `steps`.

Q11. What does the `post` block do?
Runs after the stages. Conditions: `always, success, failure, unstable, changed, fixed, aborted, cleanup`. Reports and notifications belong here.

Q12. How do you run a stage only on the main branch?
`when { branch 'main' }`, or `when { expression { env.BRANCH_NAME == 'main' } }`.

Q13. How do you handle secrets in Jenkins?
Store them in Credentials, then use `withCredentials([...])` or the `credentials()` helper in the environment block. Jenkins masks them in the log. Never commit them.

Q14. What is a Shared Library?
A separate Git repo of reusable Groovy code (`vars/`, `src/`) loaded via `@Library('qa-lib')`. It removes copy-paste across many Jenkinsfiles.

Q15. Webhook vs Poll SCM?
Webhook is push-based and instant; Git notifies Jenkins. Poll SCM is pull-based on a cron schedule and wastes cycles. Use webhook when the Git server can reach Jenkins.

Q16. Explain the `H` in Jenkins cron.
Hash. Jenkins spreads jobs across the interval using the job name so many jobs do not all fire at the same second. `H 1 * * *` = once between 01:00-01:59.

Q17. How do you parameterize a build?
Use the `parameters { }` block (choice, string, booleanParam, password). Read them with `params.NAME`.

Q18. `params` vs `env`?
`params` = values supplied when the build starts (immutable). `env` = environment variables, includes built-ins like `BUILD_NUMBER`, `JOB_NAME`, `WORKSPACE`, `BUILD_URL`.

Q19. How do you trigger job B after job A?
`build job: 'B', wait: false, parameters: [string(name:'ENV', value:'qa')]`, or an `upstream()` trigger in job B.

Q20. How do you run Selenium tests from Jenkins?
Checkout the repo, run `mvn clean test -DsuiteXmlFile=... -Dbrowser=...` on an agent that has the browser (or points to a Grid), run headless, publish JUnit/Allure in `post { always }`.

Q21. Build passes but tests failed. Why?

Either `maven.test.failure.ignore=true`, or the `sh` step exit code was swallowed, or no `junit` step exists to evaluate results.

Q22. FAILURE vs UNSTABLE vs ABORTED?

FAILURE = the step returned a non-zero exit code (compile error, script failure). UNSTABLE = build ran but tests failed (set by the junit step). ABORTED = cancelled or timed out.

Q23. How do you make Jenkins UNSTABLE instead of FAILURE on test failures?

Let Maven not fail the step (`-Dmaven.test.failure.ignore=true` or `catchError`), and let the `junit` step mark it UNSTABLE from the XML results.

Q24. How do you achieve parallel execution?

Jenkins level: `parallel { }` across agents. Test level: TestNG `parallel` + `thread-count` with a ThreadLocal WebDriver.

Q25. How do you publish Allure reports?

Add `allure-testng` + aspectjweaver, install the Allure plugin, and call `allure results: [[path: 'target/allure-results']]` in `post { always }`.

Q26. How do you send email only to the person who broke the build?

`emailext recipientProviders: [culprits(), developers()]` inside `post { failure }`.

Q27. What is Blue Ocean?

A modern Jenkins UI that visualises pipeline stages and makes failures easy to locate. Nice-to-have, not required.

Q28. How do you back up and restore Jenkins?

Copy JENKINS_HOME (or use the ThinBackup plugin). Restore by placing it back and starting Jenkins with matching plugin versions.

Q29. How do you scale Jenkins for 200 tests across 5 browsers?

Add labelled agents (or dynamic Kubernetes/Docker agents), split the suite with Jenkins `parallel`, run browsers on a Selenium Grid, and keep the controller free of builds.

Q30. Your nightly regression is flaky. What do you do?

Quantify: which tests, how often (Allure history). Quarantine flaky tests into a separate suite so the main gate is trustworthy. Fix root causes: implicit waits, shared test data, test order dependency, unstable environment. Never add retries as the first fix.

18. Quick Revision Cheat Sheet

Declarative skeleton - memorise this

```
pipeline {
  agent { label 'linux' }
  tools { jdk 'jdk17'; maven 'maven3' }
  parameters  { choice(name:'ENV', choices:['qa','uat']) }
  environment { BASE_URL = "https://${params.ENV}.acme.com" }
  options     { timeout(time:60, unit:'MINUTES'); buildDiscarder(logRotator(numToKeepStr:'20')) }
  triggers    { cron('H 1 * * 1-5') }
  stages { stage('Test') { steps { sh 'mvn -B clean test' } } }
  post   { always { junit '**/surefire-reports/*.xml' } failure { emailext(...) } }
}
```

Key steps

Step	Use
`sh` / `bat`	Run Linux / Windows command
`checkout scm`	Clone the branch that triggered the build
`cleanWs()`	Wipe the workspace
`junit '...xml'`	Publish test results, sets UNSTABLE
`allure results:[[path:'...']]`	Publish Allure report
`publishHTML(...)`	Publish any HTML report
`archiveArtifacts '...'`	Keep screenshots / jars
`emailext ...`	Rich email notification
`withCredentials([...])`	Inject a masked secret
`input message:'Approve?'`	Manual gate before PROD
`build job:'x'`	Trigger a downstream job
`parallel { }`	Run stages at the same time
`retry(2) { }` / `timeout(...) { }`	Resilience wrappers
`catchError(buildResult:'UNSTABLE')`	Continue after a failure

Built-in environment variables

```
BUILD_NUMBER   JOB_NAME     BUILD_URL      WORKSPACE
BRANCH_NAME    GIT_COMMIT   GIT_BRANCH     NODE_NAME     EXECUTOR_NUMBER
```

Command quick list

Git	Maven
`git clone / checkout -b`	`mvn clean`
`git pull --rebase`	`mvn compile`
`git add . ; git commit -m`	`mvn test`
`git push origin <branch>`	`mvn test -DsuiteXmlFile=smoke.xml`
`git log --oneline -10`	`mvn test -Dgroups=smoke`
`git revert <sha>`	`mvn package / install`
`git stash / stash pop`	`mvn -B -U clean verify`
`git tag -a v1.0 -m 'rel'`	`mvn dependency:tree`

Build status meanings

Status	Meaning	Typical cause
SUCCESS	Everything passed	-

Status	Meaning	Typical cause
UNSTABLE	Build ran, tests failed	junit step found failures
FAILURE	A step exited non-zero	Compile error, script error
ABORTED	Stopped	Timeout or manual cancel
NOT_BUILT	Stage skipped	`when` condition false

Common interview tips

- **Speak in flow, not features.** Start every answer with 'a commit comes in, then...'. Interviewers want to hear you understand the pipeline, not that you memorised plugin names.
- **Always mention the quality gate.** Say where your tests block a bad build.
- **Give a number.** 'Regression went from 3 hours to 52 minutes with 4 parallel branches' beats 'we used parallel execution'.
- **Know one failure story deeply.** Every interviewer asks 'passes locally, fails in Jenkins'. Have the Chrome headless / shm / driver-version answer ready.
- **Use current terminology:** controller and agent, not master and slave.
- **Do not pretend.** If you have not used Kubernetes agents, say 'I have not, but the concept is dynamic agents created per build'. Honest + conceptual beats bluffing.
- **Have your Jenkinsfile ready.** Be able to draw the declarative skeleton on a whiteboard in 60 seconds: pipeline, agent, tools, parameters, environment, stages, post.
- **Close with ownership.** 'I own the QA gate: smoke on every PR, regression nightly, Allure published always, mail on failure and on fixed.'

The one sentence to leave them with: "I do not just write Selenium tests - I own the quality gate that decides whether a build is allowed to reach production."