# BONUS SECTION: Interview Tips, Troubleshooting & Enterprise Examples

## BONUS 1 — Interview Perspective

### Key Topics Interviewers Ask

**Git & GitHub Questions:**

| Question | Key Points to Mention |
|---|---|
| Explain Git workflow | Clone → Branch → Code → Commit → Push → PR → Merge |
| Difference between merge and rebase | Merge preserves history, rebase creates linear history |
| How to resolve conflicts | Pull latest, open files, resolve markers, commit |
| What is a Pull Request | Code review mechanism, CI runs, approval required |
| Why use branching | Isolation, parallel work, safe experimentation |

**CI/CD Questions:**

| Question | Key Points to Mention |
|---|---|
| What is CI/CD | Continuous Integration & Delivery, automated pipelines |
| Benefits of CI/CD | Early bug detection, consistent testing, faster releases |
| Explain GitHub Actions | YAML-based workflows, triggers, jobs, steps, runners |
| How to handle secrets | GitHub Secrets, environment variables, never hardcode |
| What happens when tests fail in CI | Notification sent, PR blocked, fix required |

**Framework Questions:**

| Question | Key Points to Mention |
|---|---|
| Explain Page Object Model | Separate locators from tests, maintainability, reusability |
| How to handle flaky tests | Retries, proper waits, stable locators, investigation |
| Data-driven testing | External data files, parameterization, multiple scenarios |
| Parallel execution | Workers, sharding, independent tests |
| Reporting | HTML reporter, artifacts, CI integration |

## Do's and Don'ts in Interviews

**Technical Demonstration:**

| ✅ Do | ❌ Don't |
|---|---|
| Explain your thought process | Just give one-word answers |
| Mention real-world experience | Only speak theoretically |
| Acknowledge what you don't know | Pretend to know everything |
| Use proper terminology | Use vague language |
| Give specific examples | Be too generic |

**When Asked About Your Framework:**

```
GOOD ANSWER STRUCTURE:

1. Start with architecture overview
   "Our framework uses Page Object Model with..."

2. Explain key components
   "We have separate folders for pages, tests, and utilities..."

3. Mention CI/CD integration
   "Tests run automatically on every PR using GitHub Actions..."

4. Highlight best practices
   "We enforce code reviews, use data-driven testing..."

5. Share a challenge you solved
   "We had flaky tests, so I implemented..."
```

# BONUS 2 — Troubleshooting Tables

## Git Troubleshooting

| Error Message | Cause | Solution |
|---|---|---|
| `fatal: not a git repository` | Not in git folder | `cd` to correct folder or `git init` |
| `error: failed to push` | Remote has new commits | `git pull` first, then push |
| `CONFLICT in file.ts` | Same lines changed | Resolve conflicts manually |
| `Your branch is behind` | Need to pull updates | `git pull origin develop` |
| `Permission denied` | Auth issue | Check SSH keys or use HTTPS |
| `detached HEAD` | Not on a branch | `git checkout branch-name` |

## Pipeline Troubleshooting

| Error | Cause | Solution |
|---|---|---|
| `npm ci failed` | Missing lock file | Commit `package-lock.json` |

| Error | Cause | Solution |
|---|---|---|
| `playwright install failed` | Missing deps | Use `--with-deps` flag |
| `Timeout 60000ms exceeded` | Test too slow | Increase timeout or fix test |
| `Browser closed unexpectedly` | Memory issue | Reduce parallelism |
| `Secret not found` | Not configured | Add in repository settings |
| `Artifact upload failed` | Path doesn't exist | Check directory path |

## Test Troubleshooting

| Issue | Cause | Solution |
|---|---|---|
| Element not found | Locator changed | Update locator |
| Click intercepted | Element covered | Scroll to element first |
| Test timeout | Slow page/network | Increase timeout, check network |
| Random failures | Race condition | Add proper waits |
| Works locally, fails in CI | Environment diff | Check headless mode, viewport |

# BONUS 3 — Do's and Don'ts

## Code Writing

| ✅ Do | ❌ Don't |
|---|---|
| Use data-testid selectors | Use fragile XPath |
| Externalize test data | Hardcode values |
| Write independent tests | Create test dependencies |
| Use page objects | Write everything in test file |
| Add meaningful assertions | Skip verification |

| ✅ Do | ❌ Don't |
|---|---|
| Handle expected failures | Let tests fail silently |

## Git Usage

| ✅ Do | ❌ Don't |
|---|---|
| Pull before pushing | Force push to shared branches |
| Write clear commit messages | Use vague messages like "fix" |
| Create focused branches | Put multiple features in one branch |
| Review PR feedback promptly | Ignore review comments |
| Delete merged branches | Leave stale branches |

## CI/CD

| ✅ Do | ❌ Don't |
|---|---|
| Fix broken builds immediately | Leave pipeline red |
| Use secrets for credentials | Hardcode passwords |
| Monitor test results | Ignore CI output |
| Keep pipelines fast | Add unnecessary steps |
| Upload reports as artifacts | Lose test evidence |

## Team Collaboration

| ✅ Do | ❌ Don't |
|---|---|
| Communicate blockers early | Wait until deadline |
| Help teammates debug | Blame without helping |
| Share knowledge | Keep solutions to yourself |
| Document your work | Leave others guessing |

| ✅ Do | ❌ Don't |
| --- | --- |
| Follow agreed standards | Create your own rules |

# BONUS 4 — Real Enterprise Examples

## Example 1: E-Commerce Test Suite Structure

```
ecommerce-tests/
├── .github/
│   └── workflows/
│       ├── smoke.yml           # Quick tests on every PR
│       ├── regression.yml      # Full suite nightly
│       └── production.yml      # Production monitoring
│
├── tests/
│   ├── cart/
│   │   ├── add-to-cart.spec.ts
│   │   ├── cart-quantity.spec.ts
│   │   └── cart-removal.spec.ts
│   ├── checkout/
│   │   ├── guest-checkout.spec.ts
│   │   ├── member-checkout.spec.ts
│   │   └── payment-methods.spec.ts
│   └── search/
│       ├── product-search.spec.ts
│       └── filter-products.spec.ts
│
├── pages/
│   ├── CartPage.ts
│   ├── CheckoutPage.ts
│   ├── ProductPage.ts
│   └── SearchPage.ts
│
└── test-data/
    ├── products.json
    ├── users.json
    └── addresses.json
```

# Example 2: Multi-Environment Pipeline

```yaml
# .github/workflows/multi-env.yml
name: Multi-Environment Tests

on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Environment'
        required: true
        type: choice
        options:
          - dev
          - qa
          - staging
          - prod

jobs:
  test:
    runs-on: ubuntu-latest
    environment: ${{ inputs.environment }}

    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-node@v4
        with:
          node-version: 20

      - run: npm ci
      - run: npx playwright install --with-deps

      - name: Run tests against ${{ inputs.environment }}
        run: npx playwright test
        env:
          BASE_URL: ${{ vars.BASE_URL }}
          API_URL: ${{ vars.API_URL }}
          TEST_USER: ${{ secrets.TEST_USER }}
          TEST_PASS: ${{ secrets.TEST_PASS }}

      - uses: actions/upload-artifact@v4
        if: always()
```

```
      with:
        name: ${{ inputs.environment }}-report
        path: playwright-report/
```

## Example 3: Team Workflow Scenario

**Scenario:** Adding new checkout tests

```
DAY 1 - Monday

Engineer A (you):
├── 09:00 - Pull latest code
│   git checkout develop
│   git pull origin develop
│
├── 09:15 - Create feature branch
│   git checkout -b feature/JIRA-456-checkout-tests
│
├── 09:30 - Start writing tests
│   Created CheckoutPage.ts
│   Created checkout.spec.ts
│
├── 17:00 - Run tests locally
│   npx playwright test tests/checkout/ --headed
│   3 tests pass, 1 fails
│
├── 17:30 - Fix failing test
│   Updated locator for payment button
│
└── 18:00 - Push work
    git add .
    git commit -m "feat: add checkout page tests (WIP)"
    git push origin feature/JIRA-456-checkout-tests

DAY 2 - Tuesday

├── 09:00 - Pull latest develop
│   git checkout develop
│   git pull origin develop
│   git checkout feature/JIRA-456-checkout-tests
│   git merge develop
│
├── 09:30 - Complete remaining tests
│   Added 3 more test cases
│
├── 15:00 - Final local run
│   npx playwright test tests/checkout/
│   All 7 tests pass ✅
│
├── 15:30 - Push and create PR
│   git push origin feature/JIRA-456-checkout-tests
```

```
│   Created PR on GitHub
│
├── 16:00 - CI runs
│   Pipeline started
│   Tests running...
│   ✅ All checks pass
│
└── 16:30 - Request review
    Assigned Senior Engineer as reviewer

DAY 3 - Wednesday

├── 10:00 - Reviewer leaves comments
│   "Please add test for error handling"
│   "Use data-testid instead of class"
│
├── 11:00 - Address review comments
│   Added error handling test
│   Updated locators
│
├── 11:30 - Push updates
│   git add .
│   git commit -m "fix: address review comments"
│   git push origin feature/JIRA-456-checkout-tests
│
├── 14:00 - Review approved
│   Reviewer approved PR ✅
│
├── 14:15 - Merge to develop
│   Clicked "Squash and merge"
│   All 7 tests merged to develop
│
└── 14:30 - Cleanup
    git checkout develop
    git pull origin develop
    git branch -d feature/JIRA-456-checkout-tests

RESULT:
├── 7 new checkout tests added
├── CI pipeline passing
├── Code reviewed and approved
└── Branch cleaned up
```

# Quick Reference Card

```
┌─────────────────────────────────────────────────────────────┐
│              PLAYWRIGHT ENTERPRISE QUICK REFERENCE          │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ▶ DAILY WORKFLOW                                           │
│    git pull → branch → code → test → commit → push → PR     │
│                                                             │
│  ▶ ESSENTIAL COMMANDS                                       │
│    npx playwright test           # Run all tests            │
│    npx playwright test --headed  # See browser              │
│    npx playwright test --debug   # Debug mode               │
│    npx playwright show-report    # View report              │
│                                                             │
│  ▶ GIT COMMANDS                                             │
│    git checkout -b feature/name  # Create branch            │
│    git add . && git commit -m "" # Stage and commit         │
│    git push origin branch        # Push to GitHub           │
│                                                             │
│  ▶ TEST TAGS                                                │
│    @smoke      - Quick sanity tests                         │
│    @regression - Full test suite                            │
│    @critical   - Must-pass tests                            │
│                                                             │
│  ▶ COMMIT PREFIXES                                          │
│    feat:     - New feature                                  │
│    fix:      - Bug fix                                      │
│    refactor: - Code improvement                             │
│    docs:     - Documentation                                │
│                                                             │
│  ▶ BRANCH NAMING                                            │
│    feature/JIRA-123-description                             │
│    bugfix/JIRA-456-fix-name                                 │
│    hotfix/critical-issue                                    │
│                                                             │
│  ▶ PIPELINE STATUS                                          │
│    ✅ Green = Ready to merge                                │
│    ❌ Red   = Fix before merging                            │
│    🟡 Yellow = Running                                      │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

# Document Information

| Item | Details |
|---|---|
| **Created** | December 31, 2024 |
| **Purpose** | Enterprise onboarding & training |
| **Audience** | Automation engineers (all levels) |
| **Framework** | Playwright + TypeScript |
| **Scope** | Complete workflow from basics to enterprise practices |

*End of Enterprise Automation Handbook*