Advantages of TestNG over JUnit explain in detail

TestNG is designed to overcome many limitations of JUnit, especially for automation and end-to-end testing. TestNG gives more control over how tests are organized, run, and reported than JUnit.

Richer annotations and configuration

TestNG provides more lifecycle annotations than JUnit, such as @BeforeSuite, @BeforeTest, @BeforeClass, @BeforeMethod, and their matching @After... versions.
This allows setup and teardown at suite, test, class, and method level, which is very useful in Selenium frameworks (e.g., open browser once per suite, login once per class, clear data per method).

TestNG also supports configuration through an XML file (testng.xml), where you can define which classes, methods, or groups to run, and in what order.
JUnit generally relies on annotations and build tool configurations, so complex suite definitions are less centralized and flexible than in TestNG.

Data-driven testing with @DataProvider

TestNG has built-in data-driven testing using @DataProvider, which lets you run the same test method multiple times with different input sets supplied by a method returning Object[][] or similar structures.
This is especially powerful for Selenium tests where you want to test many username/password pairs, search terms, or form combinations without duplicating test code.

JUnit can do parameterized tests, but typically with more constraints and boilerplate; TestNG's @DataProvider is more flexible (multiple parameters, dynamic data sources like Excel or databases, etc.).

Parallel execution and performance

TestNG supports native parallel execution of tests at different levels (methods, classes, tests) configured in testng.xml.
This can significantly reduce execution time for large suites, for example when running many Selenium tests in parallel across browsers or environments.

JUnit 5 can also run tests in parallel, but it requires more configuration and is less straightforward than TestNG's built-in suite-level parallel settings.

Dependencies, grouping, and ordering

TestNG allows dependency testing using dependsOnMethods and dependsOnGroups, so you can specify that one test should run only if another test passes.
This is very useful for flows like: login test must pass before running add-to-cart or checkout tests in a UI automation suite.

You can group tests using groups and then include or exclude groups via testng.xml (e.g., smoke, regression, sanity, api, ui).
JUnit does not natively provide the same level of grouping and dependency control, so managing complex suites is harder.

Better suited for large automation frameworks

TestNG generates detailed HTML and XML reports (pass/fail/skip, time, stack traces) out of the box, which integrates nicely into automation dashboards.
JUnit usually needs plugins or external reporting tools to reach similar report quality.

Because of annotations, parallelism, data providers, dependencies, grouping, and reports, TestNG is often preferred over JUnit for Selenium and large end-to-end automation projects, while JUnit is often chosen for simple unit testing.