Command-Based Programming Explained (Text Only)

This document explains how our FRC robot program works using WPILib command-based programming. It is written for high school students and uses our actual code structure (RobotContainer, ShooterCommand, and the Intake commands) so the behavior on the dashboard makes sense. The goal is to understand not only what the code does, but why it is structured this way and how the same approach will be used on the competition robot.

1) The Big Picture: Subsystems, Commands, and the Scheduler

In command-based programming, we separate the robot into three ideas: subsystems, commands, and the scheduler. A subsystem is a hardware group like the shooter or intake. A command is a unit of robot behavior like "spin the shooter" or "run the intake." The scheduler runs every 20 ms and decides which commands should start, run, or end.

This separation makes our code easier to read, test, and expand. Instead of one giant method with lots of if-statements, we have many small commands that each do one thing. The scheduler handles the timing and prevents conflicting commands from running on the same subsystem.

2) Where This Shows Up in Our Code

In RobotContainer.java, we create our subsystems and define the button bindings. This is the control center where we say which button runs which command. We also assign default commands to subsystems. In our project, the intake subsystems have default commands, and the shooter command is triggered by the B button.

- RobotContainer.java: creates subsystems and connects inputs to commands.
- ShooterCommand.java: runs the shooter motor and logs counters to SmartDashboard.
- IntakeFalconCommand.java and IntakeSparkCommand.java: default commands for intake subsystems.

3) The Command Life Cycle

Every command has the same life cycle. The scheduler calls these methods in order:

- initialize(): runs once when the command is first scheduled.
- execute(): runs every scheduler loop while the command is scheduled.
- end(boolean interrupted): runs once when the command stops, either normally or by interruption.
- isFinished(): returns true when the command should stop on its own.

In ShooterCommand.java, we use SmartDashboard counters for initialize, execute, and end to show exactly how many times each part runs. That makes the life cycle visible and concrete.

4) Default Commands vs Non-Default Commands

Default commands are special commands assigned to subsystems. They automatically run when no other command is using that subsystem. This means the subsystem always has a command controlling it. On our robot, the intake subsystems each have a default command. That command starts when the robot is enabled and keeps running unless another command needs the intake.

Non-default commands only run when we schedule them with a button or trigger. Our shooter command is non-default. It starts when the B button is held and stops when the button is released. Because the shooter command isFinished() returns false, the only way it ends is through cancellation when the trigger becomes false.

5) Why the Dashboard Counters Look the Way They Do

The counters prove the behavior of default and non-default commands:

- Default intake commands: InitializeCount is usually 1, ExecuteCount becomes very large, and EndCount stays low because the command almost never ends.
- Shooter command: InitializeCount and EndCount increase each time you press and release the button,

Command-Based Programming Explained (Text Only)

ExecuteCount only increases while the button is held.
- cmdCancelled increases when the button is released because whileTrue() cancels the command.

This is the key lesson: default commands are "always-on" background behaviors, while non-default commands are "on-demand" behaviors tied to a trigger.

6) How the Scheduler Prevents Conflicts

Each command declares which subsystem it requires. The scheduler uses this to prevent two commands from controlling the same subsystem at the same time. If a new command is scheduled that needs a subsystem already in use, the scheduler cancels the old command (calling end(true)) and gives control to the new one. This keeps the robot safe and predictable.

This is why default commands are so useful: when a temporary command ends, the scheduler automatically returns control to the default command without any extra code. This makes the robot feel responsive and consistent to drivers.

7) How This Scales to the Competition Robot

The exact same structure will be used on the competition robot. The drivebase will have a default command that reads the joysticks. The shooter, intake, and scoring mechanisms will have button commands for specific actions like "spin up," "shoot," or "score." The scheduler guarantees that only one command runs each subsystem at a time and returns to the default behavior when the action finishes.

This approach makes it easy to add features. Want a new command for a special maneuver? Create a new command and bind it to a button. The scheduler will handle the switching automatically. This is also why command-based programming is standard in FRC: it keeps large robot programs organized and reliable.

8) Key Takeaways

- Commands define actions; subsystems define hardware; the scheduler runs everything.
- Default commands run when nothing else is using a subsystem.
- Button commands run only when scheduled by a trigger.
- end(true) means the command was interrupted (like a button release or another command).
- The counters in our ShooterCommand and Intake commands make the system visible and easy to explain.