Speaker Notes – *Terminal Coding Agents: Claude Code, Gemini CLI & Beyond*

Slide 1 – Title & Welcome

- **Opening line**: “Good morning everyone. Today we’ll explore the new generation of *terminal coding agents*—tools that can read, write and run code right from your shell.”
- Briefly state why audience (AI researchers) should care: autonomy, emergent behaviors, research frontiers.
- Logistics: mention slide controls and agenda.

Slide 2 – What Are Terminal Coding Agents?

- Define: CLI wrappers around LLMs with direct shell tool access.
- Emphasize difference from IDE plugins: no GUI, script-friendly, integrates with Unix philosophy.
- Mention exemplars (Claude Code, Gemini CLI, OpenCode, Amp) and note local-code privacy guarantee in Claude.
- **Hook question**: “How many here have piped a stack-trace into an LLM this month?” (show hands).

Slide 3 – Why They Matter – Impact & Current State

- Shift of bottleneck from typing to *spec quality*.
- Cite anecdotal gains: 30-70 % faster PR merges, solopreneur MVPs in a weekend.
- Acknowledge pain points: prompt churn, trust gaps—sets up later best-practices slides.
- Point out large context (≥100 K tokens) enabling repo-scale reasoning.

Slide 4 – Evolution of AI Coding Agents

- Walk timeline: Gen 1 autocomplete → Gen 4 terminal agents today.
- **Visual aid**: point to timeline bullets; stress each generation’s “unit of work”.
- Forecast Gen 5: multi-agent orchestration; segue into future slide.

Slide 5 – Interactive vs Unix Pipeline Modes

- Live-coding idea: open terminal, run `claude`, ask a quick Q&A.
- Then show one-liner pipeline (`cat error.log | claude …`) to illustrate batch usage.

- Highlight CI integration possibilities.

Slide 6 – Installation & First Run

- Show commands: `npm i -g @anthropic-ai/claude-code`, `npx @google/gemini-cli`.
- Mention auth/login step; note Gemini's free quota and Claude's Pro access.
- Recommend starting session in repo root, ask broad questions to preload context.

Slide 7 – Context & Memory Engineering

- Explain **CLAUDE.md** hierarchy: global, project, nested.
- Best practice: concise bullet rules; avoid dumping full design docs.
- Warn about "context poisoning" and encourage use of `@file` references.
- Tip: after every agent misstep, encode lesson into CLAUDE.md.

Slide 8 – Workflow Basics – Plan → Code → Verify

- Demonstrate asking for a plan first; prevents runaway diffs.
- Loopback workflow: agent edits → runs tests → fixes.
- Encourage treating agent like smart junior dev; review diffs incrementally.

Slide 9 – Safety, Control & Versioning

- Stress importance of Git branches; mention `permissions.json` for whitelisting.
- Show Esc-interrupt trick; recommend sandbox credentials.
- Quick story of agent deleting /tmp and how Git saved the day.

Slide 10 – Advanced Features & Parallelism

- Show sample custom slash command (`/deploy`).
- Describe MCP integration: plugging a Puppeteer headless browser for visual tests.
- Multi-boxing: tmux panes running two Claude sessions on different worktrees; speed gains.

Slide 11 – Case Study: Bug Fix + Feature

- Step through: identify failing module → agent patch → add email verification.
- Emphasize human checkpoints: approving plan, reviewing PR.
- Outcome metrics: time saved, test coverage added.

Slide 12 – Challenges in the Agent Era

- List main issues: spec ambiguity, review drag, merge conflicts, cost.
- Suggest mitigations: clearer specs, smaller diffs, trunk-based dev, cost monitors.
- Pose question to audience: “What new research directions can address trust calibration for agent-generated code?”

Slide 13 – Outlook – The Future

- Developers as “software conductors” orchestrating agents.
- Spec-driven stacks: intent → code → tests; code becomes commodity.
- Multi-agent swarms and overnight autonomous work.
- Re-affirm human roles: ethical judgment, prioritization, design.

Slide 14 – Resources & Q&A

- Point audience to docs & repos (Claude Code, Gemini CLI, OpenCode, Amp).
- Invite experimentation: “Try piping your next failing test log into a terminal agent.”
- Open floor for questions; encourage sharing of personal experiences.

Timing & Pacing Tips

- Aim ~2 min per content slide → ~25 min total.
- Keep demos < 90 s to maintain flow.
- Check audience faces mid-presentation; if confusion spotted, pause for clarifying question.

DEMO Set-Up Checklist (if live)

1. Fresh repo clone with failing tests.
2. Claude Code installed & logged in.
3. Sample error log ready for pipeline demo.
4. Screen zoom 125 % for readability.

End of speaker notes – iterate as needed!