# What is Fabric?

**Core Concept**

- Command-line framework that brings LLM power to your terminal
- Think “ChatGPT for your command line”
- Open-source and extensible

**Key Features**

- Run predefined or custom LLM patterns
- Process text from files, URLs, or command output
- Support for multiple LLM providers
- Easy integration with existing CLI tools

# Fabric Use Cases & Benefits

**Use Cases**

- Code analysis and documentation
- Log file investigation
- Content summarization
- Data extraction and transformation
- Security analysis automation

**Benefits**

- Reduces context switching between tools
- Automates repetitive tasks
- Brings AI capabilities to existing workflows

# Why Fabric Matters

**For Developers**

- Accelerates code review and documentation
- Simplifies debugging workflows
- Integrates with Git and VS Code
- Automates repetitive coding tasks

**For Security Professionals**

- Automated log analysis and threat hunting
- Quick incident response summaries
- Malware behavior analysis
- Pattern detection in large datasets

# Why Fabric Matters (continued)

**Business Impact**

- Increased productivity through automation
- Better decision-making with AI-assisted analysis
- Reduced time-to-insight for investigations
- Lower barrier to entry for AI adoption

**Key Differentiators**

- Command-line native
- Works with existing tools
- Extensible pattern system

# Demonstration

```
cat emailheader.txt | fabric -p analyze_email_headers |
tee headers.md

cat aws-flowlog.txt | fabric -p analyze_logs | tee
flowlog.md

tcpdump -r dns-remoteshell.pcap -A | fabric -p
analyze_logs -m gemini-2.0-flash-exp | tee remoteshell.md

fabric -u
https://www.trendmicro.com/en_us/research/19/g/multistage-
attack-delivers-billgates-setag-backdoor-can-turn-
elasticsearch-databases-into-ddos-botnet-zombies.html |
fabric -p analyze_malware -m gemini-2.0-flash-exp | tee
malware.md
```

# Power and Flexibility of CLI vs GUI

- **Speed & Resources**
  - Faster execution time
  - Minimal memory footprint
  - Reduced CPU usage
- **Remote Capabilities**
  - SSH access to remote systems
  - Headless server management
  - Cross-platform compatibility
- **Control & Precision**
  - Direct system interaction
  - Exact specification of parameters
- **Automation Ready**
  - Native scripting support
  - Reproducible commands

# Automation & Scripting Benefits

- **Command Chaining**

```
# Example: Find large log files and analyze with Fabric
find /var/log -size +10M | xargs fabric -p analyze_logs
```

- **Task Scheduling**

```
# Example: Daily security scan
0 0 * * * security_scan.sh | fabric -p summarize | mail -s "Daily Security
Scan Report" soc@example.com
```

- **Batch Processing**

```
# Example: Process multiple files
for file in *.log; do
  fabric -p analyze_logs "$file" > "${file%.log}_report.md"
done
```

- **Workflow Automation**

```
# Example: Automated deployment pipeline
git pull && make test && make deploy || send_alert "Deploy failed"
```

# Security Professional's CLI Toolkit

## Essential Tools & Examples

- **Log Analysis**

```
grep 'ERROR' auth.log | awk '{print $1,$2}' | sort | uniq -c
```

- **Network Monitoring**

```
tcpdump -i eth0 'port 443' -w capture.pcap
```

- **Incident Response**

```
# Quick filesystem changes check
find / -mtime -1 -ls
```

- **Process Investigation**

```
# Check for suspicious processes
ps aux | grep -i '[d]aemon\|[s]erver' | sort -rk 3,3
```

- **File Integrity**

```
# Generate & compare checksums
find /etc -type f -exec md5sum {} \; > current_sums.txt
diff current_sums.txt baseline_sums.txt
```

# Essential Linux Commands

## Basic Text Operations

- **cat**: Display file contents

```
cat file.txt
```

- **head/tail**: View start/end of files

```
head -n 5 file.txt     # First 5 lines
tail -f log.txt        # Follow log updates
```

- **grep**: Search text patterns

```
grep "error" log.txt   # Find "error" in file
```

- **echo**: Print text

```
echo "Hello World"     # Display text
```

- **tee**: Read/write streams

```
echo "test" | tee output.txt   # Display and save
```

**These commands form the foundation of text processing in Linux and are essential for working with Fabric's input/output streams.**

# File Redirection Basics

- **Standard Input (stdin):** < or 0<

```
fabric -p ai < input.txt
```

- **Standard Output (stdout):** > or 1>

```
fabric -p summarize > output.txt      # Overwrite
fabric -p summarize >> output.txt     # Append
```

- **Standard Error (stderr):** 2>

```
fabric -p not_a_pattern 2> errors.log
```

**Common Patterns**

- **Combine stdout and stderr:**

```
fabric -p ai > all.log 2>&1
```

- **Discard output:**

```
fabric -p test > /dev/null
```

**Remember: Single > overwrites, double >> appends**

# Markdown Basics

Markdown is a lightweight markup language for formatting text documents.

## Common Syntax

- **Bold text** using `**double asterisks**`
- *Italic text* using `*single asterisks*`
- Create lists with – or `1.` for numbered lists
- Links using `[text](url)`
- Headers with # (1-6 #'s for different levels)

## Code

Use backticks (`) for inline code and ```` for code blocks

# VS Code & Fabric Integration

- VS Code is a powerful IDE that supports Fabric I/O
- Not that you have to install VS Code, but it's a good IDE
- Especially nice for working with Markdown files

**Helpful VS Code Extensions**

- **Markdown Preview Enhanced**: For viewing Markdown files
- **Markdown PDF**: Convert Markdown to PDF
- **vscode-pdf**: For viewing PDF files

**Chrome Extension**

- Markdown Viewer Extension: https://github.com/simov/markdown-viewer

# The Workshop Development Environment

## Code-Server Access

- Each student has received:
  - Personal URL (e.g., `https://workshop-XX.example.com`)
  - Unique password for authentication
- Code-server provides VS Code in your browser
- No local installation needed

## Pre-configured Environment

Your VM includes:

- Fabric framework pre-installed
- Required extensions loaded
- Pattern directories configured
- Test data available

# Accessing Your Development Environment

**Getting Started**

1. Open your assigned URL in any browser
2. Enter your provided password
3. You'll see a familiar VS Code interface
4. Terminal is ready with Fabric commands

Take a moment and do this now.

# Hands-on Exercise

**Running Fabric in VS Code Terminal**

1. Open integrated terminal (`Ctrl+`)
2. Basic command structure:

```
# Look at the Help System
fabric --help

# Run a Pattern
fabric --pattern hello_world

# Stream output to console
fabric --pattern hello_world --stream

# or this way for short
fabric -s -p hello_world
```

# Commands to Run for Yourself

```
fabric -h # View the fabric help

fabric -L # View the list of models

fabric -l # View the list of patterns

fabric -p summarize --dry-run # View the summarize pattern

cat input.md | fabric -p summarize # Summarize the
input.md file
```

# Introduction to Prompt Engineering

**Key Concepts**

- Crafting effective prompts for LLMs
- Understanding context and specificity
- Balancing precision with flexibility
- Iterative refinement of prompts
- Handling edge cases and errors
- Understanding Temperature

**Resources**

- Best Practices for Prompt Engineering with the OpenAI API
- Prompt Engineering Guide

# Prompt Examples

**Bad Prompts**

- “analyze this” or “what’s wrong with this code?”
- “fix it” or “make it better”
- “check for security issues”

**Good Prompts**

- “Analyze this Python code for potential security vulnerabilities, focusing on input validation and SQL injection risks. Format findings as a bulleted list with severity levels.”
- “Review this Apache access log for suspicious patterns indicating potential intrusion attempts. Group findings by IP address and timestamp.”
- “Examine this network traffic capture for signs of data exfiltration, particularly focusing on unusual DNS queries and HTTPS patterns. Highlight any IPs or domains that require investigation.”

# Understanding Temperature in LLMs

**What is Temperature?**

- A parameter that controls response randomness
- Scale from 0.0 to 1.0
- Higher values = more creative/random
- Lower values = more focused/deterministic

**Temperature Guidelines**

- 0.0: Best for factual responses, code analysis, security scanning, pattern matching
- 0.7: Good for creative writing, brainstorming, general conversation
- 1.0: Suitable for maximum creativity, story generation, exploring alternatives

**Normally, you can just leave the default value (0.7)**

# LLM Hallucinations

**What are Hallucinations?**

- When LLMs generate false or made-up information
- Can appear convincing but be completely incorrect
- A significant challenge in AI safety and reliability

**Common Types of Hallucinations**

- Fabricating facts, statistics, or references
- Creating non-existent citations or sources
- Inventing technical details or procedures
- Mixing up or combining unrelated information

# Mitigating Hallucinations

- Use lower temperature settings (0.0-0.3)
- Always verify critical information
- Cross-reference with trusted sources
- Be especially careful with:
  - Technical specifications
  - Historical dates and facts
  - Citations and references
  - Security-related information

**Best Practices**

- Treat LLM outputs as suggestions, not facts
- Implement human verification for critical tasks
- Use system prompts that emphasize accuracy
- Consider using multiple LLMs for cross-validation

# Understanding the `context` Parameter

- Context files tell the AI how to respond to best meet your needs.
- Fabric expects the files to be in the `~/.config/fabric/contexts` directory.
- Use `-C` or `--context` to specify the context file.
- use `-x` to list the context files.

**Move the files and try them out:**

```
cp context-*.md ~/.config/fabric/contexts/
```

```
echo "explain the CIA Traid" | fabric -C context-expert.md
-p raw_query
```

```
echo "explain the CIA Traid" | fabric -C context-
layperson.md -p raw_query
```

# Understanding Fabric Sessions

- When you use the `--session=` parameter, Fabric will use the session file to store the conversation history.
- This will either create a new session or continue an existing session.
- The session file is stored in the `~/.config/fabric/sessions` directory.
- Use `-X` or `--listsessions` to list the sessions.
- Use `-W` or `--wipesession=` to delete a session.
- To print the session, use `--printsession`.
- Output the entire session to the output file using `--output-session`.
- Specify the output file using `-o` or `--output`.

# Understanding Fabric Patterns

**What are Fabric Patterns?**

- Pre-defined prompt templates that leverage LLMs for specific tasks
- Building blocks for automating common workflows
- Can be customized and extended for specific needs
- Designed for reusability and consistency

**Let’sExamine Some Patterns**

- summarize
- extract_wisdom
- analyze_logs

# Using Fabric Patterns Effectively

**Best Practices**

- Examine the contents of the pattern
- Choose the right pattern for your use case
- Understand pattern inputs and outputs
- Test patterns with sample data first
- Document any customizations you make
- Share successful patterns with the community

# Git & GitHub Essentials

## Version Control Basics

- Git tracks changes in your code over time
- Enables collaboration between developers
- Maintains history of all modifications
- Track who changed what and when
- Revert to previous versions

## Key Git Commands

```
git clone     # Copy a repository
git add       # Stage changes
git commit    # Save changes
git push      # Upload to GitHub
git pull      # Download updates
```

# The Fabric Repository

## Repository Overview

github.com/danielmiessler/fabric

## Key Components

- `/patterns`: AI prompt templates
- `README.md`: Getting started guide
- Commit History

## Community Features

- Issues: Bug reports and feature requests
- Discussions: Community interaction
- Pull Requests: Contribute improvements
- Stars: ⭐ Show support

# Understanding GitHub Codespaces

**What is Codespaces?**

- Cloud-based development environment
- Full VS Code in your browser
- Pre-configured development container
- Accessible from anywhere

**Key Benefits**

- No local setup required
- Consistent environment
- Integrated with GitHub
- Full terminal access
- Extensions pre-installed

# Setting Up Fabric in Codespaces

**Quick Start**

1. Navigate to fabric_on_codespace repository
2. Click “Code” button
3. Select “Open with Codespaces”
4. Choose machine type

**Environment Features**

- Fabric pre-installed
- Required dependencies ready
- Pattern directories configured
- Github authentication handled

NOTE: In Codespaces you must provide your own API keys for Fabric.

# Configuring Your Environment

1. Once your Codespace loads, create a `.env` file:

```
cp .env-example .env
```

2. Edit the `.env` file to add your API keys:

```
OPENAI_API_KEY=sk-...
ANTHROPIC_API_KEY=sk-...
```

# Troubleshooting Common Issues

If API calls fail:

- Verify your `.env` file exists
- Check your API keys are valid
- Ensure no extra spaces in `.env`
- Run `fabric -L` to see if the models are available

# Codespaces Best Practices

- Always use environment variables for API keys
- Do not commit your `.env` file!
- Keep your Codespace up to date (Commit any changes)

**Remember to stop your Codespace when not in use to conserve resources**

# Prompts Deep Dive: Crafting Effective Prompts

**Key Elements**

- Be clear and specific in your instructions
- Break complex tasks into smaller steps
- Include relevant context and constraints
- Use consistent formatting and structure
- Specify the desired output format

**Example**

```
# Less effective
echo "analyze this: $(cat document.txt)" | fabric -p ai

# More effective
echo "analyze this technical document and highlight security
implications: $(cat document.txt) " | fabric -p ai
```

# Understanding Context & Specificity

**Context Matters**

- Provide relevant background information
- Define technical terms and acronyms
- Specify the target audience
- Include any necessary constraints
- Set the scope of the analysis

**Example**

```
# Adding context
echo "Review this code from a GDPR compliance perspective,
focusing on data privacy requirements for EU customers:
$(cat my_script.py)" | fabric -p ai
```

# Balancing Precision & Flexibility

**Finding the Sweet Spot**

- Be precise enough to get desired results
- Leave room for LLM's capabilities
- Avoid over-constraining the response
- Allow for creative solutions
- Use guardrails when needed

**Example**

```
# Too rigid
echo "Write exactly 5 bullet points about security" | fabric -p
ai

# Better balance
echo "Write a concise security analysis focusing on key risks.
Use bullet points." | fabric -p ai
```

# Iterative Prompt Refinement

## The Refinement Process

- Start with a basic prompt and analyze the output quality
- Identify areas for improvement
- Adjust and test incrementally

## Example

```
# Initial attempt
echo "Check this log file: $(cat security.log)" | fabric -
p ai

# Refined version
echo "Analyze this log file for failed login attempts,
highlighting IP addresses and timestamp patterns: $(cat
security.log)" | fabric -p ai
```

# Handling Edge Cases & Errors

**Best Practices**

- Anticipate potential failure modes
- Include error handling instructions
- Validate input data quality
- Plan for unexpected outputs
- Use defensive prompting techniques

**Example**

```
# With error handling
echo "Review this log file for security incidents. If the
file is empty or corrupted, report the issue. If no
incidents found, explicitly state that. Log: : $(cat
security.log)" | fabric -p ai
```

# Patterns Deep Dive: Anatomy of a Fabric Pattern

Fabric patterns ARE prompts

**Core Sections**

- IDENTITY and PURPOSE: Defines the AI's role
- STEPS: Clear instructions for task completion
- OUTPUT INSTRUCTIONS: Formatting and structure rules
- INPUT: The data to be processed (typically blank)

**Why This Structure Matters**

- Creates consistent, reliable outputs
- Makes patterns reusable and maintainable
- Ensures clear communication with the LLM

# Defining Pattern Identity & Purpose

**Key Components**

- Clear role definition for the AI
- Specific responsibilities
- Scope of operations
- Success criteria
- Contextual boundaries

**Example Pattern Header**

```
# IDENTITY and PURPOSE
You are an AI security log analyzer responsible for
identifying potential security incidents in system logs.
You meticulously examine each log entry for patterns
indicating suspicious activity...
```

# Structuring Pattern Steps

**Best Practices**

- Break down complex tasks
- Use sequential, logical order
- Make steps atomic and clear
- Include validation points
- Define expected outcomes

**Example Steps Section**

```
# STEPS
- Extract relevant log entries based on timestamp
- Identify IP addresses and user agents
- Compare against known threat patterns
- Categorize severity of findings
- Format results in specified structure
```

# Defining Output Instructions

## Essential Elements

- Specify output format (Markdown, JSON, etc.)
- Define structure and hierarchy
- Include formatting rules
- Provide validation criteria
- Set quality standards

## Example Output Section

```
# OUTPUT INSTRUCTIONS
- Output must be in Markdown format
- Use H2 for main findings
- List each incident with timestamp
- Include severity rating (High/Medium/Low)
- Ensure all IPs are properly formatted
```

# Creating Complete Patterns

**Pattern Development Workflow**

1. Define the AI's role clearly
2. Break down the task into steps
3. Specify output requirements
4. Include example inputs/outputs
5. Test and refine

**Tips for Success**

- Think step-by-step
- Be explicit about requirements
- Test with various inputs
- Document pattern behavior

# Getting Hands-on with Fabric

**Key Use Cases**

- Document Analysis & Claims Extraction
- Security Log Analysis
- Incident Response
- Code Review & Documentation
- Threat Analysis & Reporting

**Why These Examples?**

- Real-world applications
- Common security workflows
- Demonstrate pattern flexibility
- Show practical value

# Document Analysis Examples

## Useful Patterns

- `analyze_claims`
- `extract_extraordinary_claims`
- `extract_insights`
- `create_cyber_summary`

## Example Usage

```
# Extract key claims from a document
cat threat_report.md | fabric -p analyze_claims

# Create a cybersecurity summary
cat advisory.md | fabric -p create_cyber_summary
```

# Security Log Analysis

## Relevant Patterns

- analyze_logs
- analyze_incident
- create_sigma_rules
- extract_poc

## Example Usage

```
# Analyze security logs
cat security.log | fabric -p analyze_logs

# Create detection rules
cat incident.json | fabric -p create_sigma_rules
```

# Threat Analysis & Reporting

## Key Patterns

- analyze_threat_report
- analyze_threat_report_trends
- create_stride_threat_model
- create_network_threat_landscape

## Example Usage

```
# Analyze a threat report
cat threat_report.md | fabric -p
analyze_threat_report_trends

# Create threat model
cat architecture.md | fabric -p create_stride_threat_model
```

# Reverse Engineering Fabric Patterns

```
fabric -l # list all patterns

ls ~/.config/fabric/patterns/ # Examine the pattern directory

fabric -p extract_wisdom --dry-run # Read the pattern's prompt

fabric -p analyze_threat_report --dry-run > analyze_threat_report.md

nano analyze_threat_report.md  # Edit the pattern
```

# Creating Custom Patterns

**Example: AWS CloudTrail Analysis Pattern**

```
# Set up pattern directory
export PATTERN_DIR="$HOME/.config/fabric/patterns"
mkdir -p $PATTERN_DIR/analyze_aws_cloudtrail

# Create the pattern
echo "Analyzing AWS CloudTrail logs that identifies
privilege escalation attempts" | fabric -p create_pattern
| tee $PATTERN_DIR/analyze_aws_cloudtrail/system.md

# Inspect the pattern
cat $PATTERN_DIR/analyze_aws_cloudtrail/system.md

# Test the pattern
cat cloudtrail.log | fabric -p analyze_aws_cloudtrail
```

# Chaining Commands to Exploit CLI Power

**The Power of Unix Philosophy**

- Each program does one thing well
- Programs work together
- Programs handle text streams as universal interface

**Understanding Subshells**

- A subshell is a child process of the current shell
- Created using `$()` or backticks `` ` ``
- Example: `echo "Today is $(date)"`
- Nested commands execute from innermost to outermost
- Useful for command substitution and complex pipelines

# Chaining Commands - Loops

**Bash For Loops**

- Iterate over lists, ranges, or command output
- Basic syntax:

```
for item in list; do
    command $item
done
```

- Examples:

```
# Loop over numbers
for i in {1..5}; do echo $i; done

# Loop over files
for file in *.txt; do cat $file; done

# Loop over command output
for user in $(who | cut -d' ' -f1); do
    echo "Hello $user"
done
```

# Hands on Exercise: Summarize a Pattern

```
# Summarize the extract_wisdom pattern
fabric p extract_wisdom --dry-run | fabric -p
summarize_prompt > output.md
cat output.md

# Use `tee` to write to a file and stdout
fabric -p extract_wisdom --dry-run | fabric -p
summarize_prompt | tee output.md
```

TIP: Try different models! (This works best with OpenAI GPT4o)

# Hands on Exercise: Summarize All Patterns

```
# Loop through all patterns
for pattern in $(fabric -l); do echo "--> "$pattern; done

for pattern in $(fabric -l); do
    echo -e "\n## "$pattern | tee -a summaries.md;
    fabric -p $pattern --dry-run | fabric -p
summarize_prompt | tee -a summaries.md;
done
```

Protip: Ask AI to explain commplicated CLI commands

Note the use of append mode `tee -a` to add to the file.

# Q&A and Wrap-Up

**Key Takeaways**

- Fabric enhances CLI workflows with LLM capabilities
- Command chaining multiplies tool effectiveness
- Integration with VS Code and GitHub streamlines development
- Security use cases demonstrate practical applications

**Questions?**

# Resources for Further Learning

**Documentation & Repositories**

- Fabric GitHub Repository
- Fabric Documentation
- VS Code Command Line Tools

**CLI Learning Resources**

- Tips for success with Command Line Interfaces using BASH
- Slice and Dice Data using grep, head, tail, cut, sort, tr, uniq and wc

- explainshell.com - Decode command-line arguments
- commandlinefu.com - Community-driven command-line tips
- ss64.com - Command line reference
- Learn Shell - Learn Shell

# Getting Help & The FabricCommunity

**Stay Connected to the Fabric Project**

- Fabric Intro Video
- Follow @danielmiessler on Twitter/X
- Learn more at danielmiessler.com/

**Contributing to Fabric**

- How to report issues or get help

- Contributing guidelines

# Workshop Feedback

**Help Us Improve**

- Complete the feedback survey
- Share your experience
- Suggest improvements
- Request future topics

Feedback Survey

**Thank You!**

Thank you for participating in the Mastering Fabric Workshop

# Contact Information

**Lucid Truth LLC**

- Website: LucidTruthTechnologies.com
- Email: ken@lucid-truth.comclear

- Twitter/X: @kennethghartman
- LinkedIn: @kennethghartman
- Subscribe to our newsletter at the bottom of the Lucid Truth Technologies Blog page.

**Workshop Materials**

- Slides and code available at: github.com/Resistor52/fabric-workshop
- Fabric Learning Environment: github.com/Resistor52/fabric-course-vm
- Fabric on Codespaces: github.com/Resistor52/fabric-on-codespaces