Getting Started with Port 42

> From simple tools to reality compiler mastery - a progressive guide to consciousness computing

Welcome to Port 42! This guide takes you from your first simple tool to advanced context-aware reality compilation. Each section builds on the previous, introducing new concepts progressively.

Quick Setup (2 minutes)

```
# 1. Install Port 42
curl -fsSL https://raw.githubusercontent.com/yourusername/port42/main/install.sh | bash

# 2. Set your API key when prompted, or:
export ANTHROPIC_API_KEY='your-key-here'

# 3. Verify everything works
port42 status
# Should show:  Daemon running on port 4242
```

Learning Path

Level 1: Basic Tool Creation (5 minutes)

Start with simple declarative tool creation:

```
# Create your first tool
port42 declare tool hello-port42 --transforms greeting,demo

# Use it immediately
hello-port42
# Output: Hello from Port 42! This is a demo greeting tool.

# See what was created
port42 ls /commands/
# Shows: hello-port42 (executable)

port42 cat /commands/hello-port42
# Shows the generated script

# Explore the virtual filesystem
port42 ls /
# Shows: commands/, tools/, memory/, by-date/, similar/
```

Success! You've just experienced **declarative reality creation** - you declared what should exist, and the reality compiler made it real.

Virtual Filesystem Preview: Notice that Port 42 presents everything through a unified virtual filesystem. The `/similar/` path (which you'll explore in Level 9) automatically discovers relationships between your tools!

Level 2: Transform-Based Capabilities (10 minutes)

Tools are defined by their **transforms** (what they do), not their names:

```
# Create tools with different capability combinations
port42 declare tool data-parser --transforms data,parse,json
port42 declare tool log-analyzer --transforms log,analyze,pattern
port42 declare tool text-formatter --transforms text,format,markdown

# Discover tools by capability (semantic search)
port42 search "json"        # Finds data-parser
port42 search "analyze"     # Finds log-analyzer
port42 search "format"      # Finds text-formatter

# Try your tools
data-parser sample.json
log-analyzer /var/log/system.log
text-formatter README.txt
```

Key Insight: Transforms make tools discoverable by capability, not just name.

Level 3: AI-Assisted Creation (15 minutes)

Move from declaration to conversation:

```
# Start an AI session
port42 possess @ai-engineer

# Now you're in conversation mode
> Create a command that converts CSV files to beautiful HTML tables

# AI will generate and materialize the tool
  Crystallizing your intention...
[Created: csv-to-html-converter]

# Exit the session
> exit

# Use your new tool
csv-to-html-converter data.csv > report.html
```

Pro Tip: Commands Work in AI Sessions

All your created commands are available within AI possession sessions:

```
port42 possess @ai-engineer

> List all my available commands
# AI will show you all your tools

> Run csv-to-html-converter on my sales data
```

```
# AI can execute your commands and discuss results

> What does my log-analyzer tool do again?
# AI can explain your tools and suggest improvements
```

Try different AI personalities: - `@ai-engineer` - Technical implementation focus - `@ai-muse` - Creative and elegant solutions
- `@ai-growth` - Business and productivity tools - `@ai-founder` - Strategic and visionary tools

Key Insight: AI agents understand context and can both create tools AND help you use existing ones.

Level 4: File References - Local Context (20 minutes)

Reference local files to give tools understanding of your project:

```
# Create a sample config file
echo '{
  "api_url": "https://api.example.com",
  "timeout": 30,
  "retry_count": 3
}' > app-config.json

# Reference it in tool creation
port42 declare tool config-validator --transforms validate,config,json \
  --ref file:./app-config.json

# The tool now understands your config structure
config-validator another-config.json
# Will validate against your specific schema

# Reference multiple files
port42 declare tool project-analyzer --transforms analyze,project \
  --ref file:./package.json \
  --ref file:./README.md \
  --ref file:./app-config.json

# Tool has full project context
project-analyzer --report
```

Key Insight: File references give tools deep understanding of your specific project structure and requirements.

Level 5: Port 42 VFS - Crystallized Knowledge (25 minutes)

Reference existing tools and Port 42's knowledge base:

```
# First, create a base tool
port42 declare tool base-processor --transforms data,process,transform
```

```
# Reference existing tools to build on them
port42 declare tool enhanced-processor --transforms process,enhance,optimize \
  --ref p42:/commands/base-processor

# Reference commands from the VFS
port42 declare tool super-tool --transforms analyze,process,output \
  --ref p42:/commands/existing-analyzer \
  --ref p42:/commands/data-processor

# Search for tools in the knowledge base
port42 declare tool smart-analyzer --transforms analyze,intelligent \
  --ref p42:/knowledge/analysis-patterns
```

Key Insight: P42 VFS lets you build on existing knowledge and tools, creating increasingly sophisticated capabilities.

Level 6: Web References - External Knowledge (30 minutes)

Reference web content for API specs, documentation, and examples:

```
# Reference API documentation
port42 declare tool github-client --transforms http,github,api \
  --ref url:https://docs.github.com/en/rest

# Reference multiple sources
port42 declare tool smart-api-client --transforms http,client,robust \
  --ref url:https://jsonapi.org/format/ \
  --ref url:https://httpbin.org/json \
  --ref file:./api-examples.md

# The tool understands the API spec and your examples
github-client repos list
smart-api-client --endpoint /users --format jsonapi
```

Key Insight: Web references let tools understand external APIs, standards, and documentation.

Level 7: Custom AI Instructions with Prompts (35 minutes)

Guide AI generation with specific instructions using the `--prompt` parameter:

```
# Basic prompt usage
port42 declare tool smart-log-analyzer --transforms analyze,logs,security \
  --prompt "Create a log analyzer that focuses on security threats, extracts IP addresses, dete

# Test the focused generation
smart-log-analyzer /var/log/auth.log
# Will specifically look for security-related patterns

# Combine prompts with references for context-aware instructions
```

```
port42 declare tool project-validator --transforms validate,lint,security \
  --ref file:./eslint.config.js \
  --ref file:./security-policy.md \
  --prompt "Build a comprehensive validator that enforces our ESLint rules, checks for security

# Advanced prompt with artifact generation
port42 declare artifact deployment-guide --artifact-type documentation \
  --ref file:./docker-compose.yml \
  --ref file:./README.md \
  --prompt "Generate a complete deployment guide that explains our Docker setup, includes troul
```

AI-Assisted Conversations with Context:

```
# Start conversation with references and ask specific questions
port42 possess @ai-engineer \
  --ref file:./architecture.md \
  --ref p42:/commands/existing-microservice \
  "Help me design a new authentication service that integrates with our existing architecture"

# Creative work with brand context
port42 possess @ai-muse \
  --ref file:./brand-guidelines.pdf \
  --ref file:./previous-campaigns.md \
  "Create a product announcement that follows our brand voice and builds on our previous messag
```

Prompt Best Practices: - Be specific about what you want the tool to do - Include quality criteria (e.g., "with detailed error messages") - Specify output format when relevant - Combine with references for maximum context

Key Insight: Prompts let you provide specific AI guidance while references provide contextual knowledge.

Level 8: Multi-Reference Intelligence (40 minutes)

Combine all reference types for maximum context:

```
# The ultimate context-aware tool
port42 declare tool intelligent-data-processor --transforms data,process,analyze,output \
  --ref file:./data-schema.json \
  --ref p42:/commands/base-processor \
  --ref url:https://json-schema.org/specification.html \
  --ref search:"data processing patterns" \
  --ref tool:existing-validator

# This tool now has:
#   Your specific data schema (file reference)
#   Existing processing logic (p42 reference)
#   JSON Schema standards (web reference)
#   Best practices knowledge (search reference)
```

```
#  Validation capabilities (tool reference)

intelligent-data-processor input.json --validate --optimize
```

Key Insight: Multiple references create tools with deep, multi-layered understanding.

Level 8: Memory and Continuity (50 minutes)

Reference previous conversations and build continuity:

```
# Start a design session
port42 possess @ai-engineer --session project-design

> I need to build a log processing system for a web application
> It should handle nginx logs, extract patterns, and generate reports

[Conversation continues... session ID: cli-1234]

#  While in the AI session, you can use existing commands:
> Show me what data-parser does again
> Run file-validator on my config.json
> List all my analysis tools

> exit

# Later, reference that design session
port42 declare tool log-processor --transforms log,process,report \
  --ref p42:/memory/cli-1234 \
  --ref file:./nginx-sample.log \
  --ref url:https://nginx.org/en/docs/http/ngx_http_log_module.html

# Tool is created with full design context
log-processor /var/log/nginx/access.log --pattern-analysis
```

AI Sessions + Existing Tools:

Your AI agents can interact with all your existing tools:

```
port42 possess @ai-muse

> What tools do I have for processing data?
# AI lists relevant tools: data-parser, csv-converter, json-formatter

> Run data-parser on the sales.csv file and explain the results
# AI executes your command and interprets the output

> Create a workflow that uses data-parser, then csv-converter, then formats with json-formatte
# AI can orchestrate multiple existing tools
```

Key Insight: Memory references create continuity between conversations and implementations,

and AI can leverage your entire tool ecosystem.

Level 9: Semantic Tool Discovery (60 minutes)

Master the automatic similarity detection and exploration system:

```
# Step 1: Create some analysis tools to demonstrate similarity detection
port42 declare tool log-analyzer --transforms logs,analysis
port42 declare tool data-analyzer --transforms data,analysis
port42 declare tool quick-analyzer --transforms quick,analysis

# Step 2: Explore automatic similarity relationships
port42 ls /similar/                    # See all tools with similarities (150+)
port42 ls /similar/log-analyzer/       # Find tools similar to log-analyzer
port42 ls /similar/data-analyzer/      # Find tools similar to data-analyzer

# Step 3: Discover cross-category relationships
port42 declare tool basic-parser --transforms parse,basic
port42 declare tool data-processor --transforms data,process
port42 ls /similar/data-processor/     # May find parsers (both process data)

# Step 4: Understand the semantic intelligence
port42 ls /similar/analyzer/           # Tools with 'analyze' find 'analysis' tools
# Results show: log-analyzer, data-analyzer, semantic-analyzer, etc.

# Step 5: Explore the mathematical precision
port42 ls /similar/log-analyzer/ | wc -l     # Count similar tools (likely 40+)
port42 ls /similar/basic-parser/ | wc -l     # Count similar tools (likely 10+)
```

Advanced Discovery Techniques:

```
# Find tools by capability without knowing names
port42 ls /similar/analyzer/           # "I need something that analyzes"
port42 ls /similar/parser/             # "I need something that parses"
port42 ls /similar/processor/          # "I need something that processes"

# Explore tool ecosystems
port42 ls /similar/security-test/      # Find all security-related tools
port42 ls /similar/view-analyzer/      # Find all viewer tools

# Quality assurance via bidirectional relationships
port42 ls /similar/A/ | grep B         # Check if A finds B
port42 ls /similar/B/ | grep A         # Check if B finds A (both should work)

# Combine with traditional discovery
port42 search "analysis"               # Text-based search
port42 ls /similar/log-analyzer/       # Similarity-based discovery
port42 ls /tools/by-transform/analysis/  # Transform-based browsing
```

Key Insights: - **Automatic Discovery**: Every tool you create is automatically analyzed

for similarity - **150+ Tool Scale**: System handles large collections with 18ms response times - **Mathematical Precision**: Uses Jaccard similarity + semantic enhancement
- **Bidirectional Relationships**: If A is similar to B, then B is similar to A - **Cross-Category Intelligence**: Parsers can find processors, analyzers can find viewers

Real-World Use Case:

```
# Scenario: You forgot the name of a tool but remember it does analysis
port42 ls /similar/analyzer/           # Shows ALL analysis tools regardless of name
# Result: log-analyzer, quick-analyzer, semantic-analyzer, test-analyzer, etc.

# Scenario: You want to find tools like an existing one
port42 ls /similar/my-existing-tool/   # Shows similar capabilities
```

The similarity system transforms Port 42 from a file browser into an intelligent capability discovery engine.

Level 9.5: Auto-Spawning and Rules Engine (65 minutes)

Port 42 has intelligent rules that automatically create related tools:

```
# Create an analysis tool
port42 declare tool log-analyzer --transforms log,analyze,patterns

#  AUTOMATIC: Rules engine detects "analysis" tool and auto-spawns viewer
#  Auto-created: view-log-analyzer (for viewing analysis results)

# Check what was automatically created
port42 ls /tools/log-analyzer/spawned/
# Shows: view-log-analyzer -> automatically created viewer tool

# Both tools are now available
log-analyzer /var/log/nginx/access.log > analysis.json
view-log-analyzer analysis.json  #  Auto-spawned viewer!
```

Rules Engine Intelligence:

```
# Any tool with "analysis" transforms gets a viewer
port42 declare tool data-analyzer --transforms data,analyze
#  Auto-spawns: view-data-analyzer

# Any tool with "process" transforms gets a viewer
port42 declare tool file-processor --transforms file,process,transform
#  Auto-spawns: view-file-processor

# Check the spawning relationships
port42 ls /tools/spawned-by/         # Global spawning index
port42 ls /tools/ancestry/           # Parent-child relationships
port42 info /tools/view-data-analyzer  # Shows parent relationship
```

Self-Organizing System:

The rules engine creates an ecosystem where tools automatically work together:

```
# Create a comprehensive analysis tool
port42 declare tool system-analyzer --transforms system,analyze,report

# Rules engine automatically creates:
#   view-system-analyzer (for viewing reports)
#   Links to related analysis tools
#   Spawning relationships in VFS

# Your tool ecosystem grows intelligently
port42 ls /tools/by-transform/analyze/    # All analysis tools
port42 search "analyze"                   # Semantic discovery finds all
```

Level 10: Advanced Patterns (Master Level)

Combine everything for powerful patterns:

```
# 1. Progressive Enhancement Pattern
# Base tool
port42 declare tool basic-analyzer --transforms analyze,basic

# Enhanced version with context
port42 declare tool smart-analyzer --transforms analyze,intelligent \
  --ref tool:basic-analyzer \
  --ref search:"analysis patterns" \
  --ref url:https://example.com/analysis-guide

# Project-specific version
port42 declare tool project-analyzer --transforms analyze,project \
  --ref tool:smart-analyzer \
  --ref file:./project-spec.md \
  --ref p42:/memory/requirements-session

# 2. Knowledge Synthesis Pattern
port42 possess @ai-engineer --session architecture

> Design a microservices monitoring system
[Create comprehensive design]
> exit

# Implement with full context
port42 declare tool service-monitor --transforms monitor,microservice,alert \
  --ref p42:/memory/architecture-session \
  --ref file:./docker-compose.yml \
  --ref url:https://prometheus.io/docs/concepts/ \
  --ref p42:/commands/base-monitor \
  --ref search:"monitoring best practices"
```

```
# 3. Continuous Evolution Pattern
# Tools that reference and improve each other over time
port42 declare tool evolved-processor --transforms process,evolve \
  --ref tool:previous-processor \
  --ref p42:/memory/feedback-session \
  --ref file:./new-requirements.md
```

Mastery Checklist

You've mastered Port 42 when you can:

- ☐ Create tools declaratively with `--transforms`
- ☐ Use AI agents for conversational creation
- ☐ **Use existing commands within AI sessions**
- ☐ **Ask AI to list and explain your tools**
- ☐ Reference local files for project context
- ☐ Reference P42 VFS for existing knowledge
- ☐ Reference web content for external specs
- ☐ Combine multiple reference types intelligently
- ☐ Use memory references for continuity
- ☐ **Understand auto-spawning and viewer tool creation**
- ☐ **Navigate spawning relationships in VFS**
- ☐ Discover tools semantically with search
- ☐ Navigate the virtual filesystem fluently
- ☐ Create sophisticated multi-layered tools
- ☐ Build progressive enhancement patterns
- ☐ Design knowledge synthesis workflows
- ☐ **Leverage the rules engine for automatic tool ecosystems**

What's Next?

For Power Users: - Explore advanced AI agent customization - Build tool ecosystems with spawning relationships - Create team knowledge sharing workflows - Design custom reality compilation patterns

For Developers: - Contribute new reference resolver types - Extend the AI agent personalities - Build integrations with external systems - Contribute to the reality compiler architecture

For Organizations: - Deploy Port 42 for team knowledge management - Create organizational tool libraries - Build custom AI agents for domain expertise - Integrate with existing development workflows

Key Principles to Remember

1. **Declare What Should Exist** - Focus on outcomes, not implementation
2. **Build with Context** - Use references to create intelligent tools
3. **Semantic Discovery** - Search by capability, not just names
4. **Progressive Enhancement** - Build on existing knowledge and tools
5. **Continuity Through Memory** - Connect conversations to implementations

6. **Intelligence Through References** - More context = smarter tools
7. **Self-Organizing Ecosystems** - Rules engine creates tool relationships automatically
8. **AI Tool Integration** - Use all your tools within AI conversations

Welcome to the reality compiler. The dolphins are listening.

Ready to dive deeper? Check out the full documentation or start experimenting with your own tool ideas!