3D Printing Course for Blind and Visually Impaired Learners

Michael Ryan Hunsaker, M.Ed., Ph.D. Updated: 2026-02-24

Book URL: https://mrhunsaker.github.io/VI_3DMake_OpenSCAD_Curriculum/

Contents

Introduction

This course teaches 3D design and digital fabrication using a fully accessible, command-line-driven toolchain centered on OpenSCAD (text-based CAD), 3DMake (non-visual build automation), and accessible editors (VS Code, Notepad++, command-line editors) with screen reader support.

The curriculum is explicitly designed for blind and visually impaired learners who use screen readers (NVDA, JAWS, VoiceOver). It eliminates GUI navigation and visual feedback in favor of keyboard-driven, text-based workflows that screen readers can fully access. Accessibility is not an add-on. It is the foundation of every tool, workflow, and lesson in this curriculum.

Curriculum Structure

Screen Reader Setup & Accessibility Fundamentals

Before choosing a command-line pathway, students learn to optimize their screen reader setup for terminal work and understand accessibility options.

What's Included:

- Screen Reader Accessibility Guide - Comprehensive NVDA & JAWS reference with terminal tips for PowerShell, CMD, and Git Bash
- Screen Reader Choice: Windows CLI - Help choosing between NVDA and JAWS
- Braille Display & Terminal Mode - Advanced accessibility setup with refreshable braille displays
- Editor Selection and Setup - Choosing and configuring an accessible text editor

Why start here? Screen reader optimization is the foundation for all command-line work. Students learn keyboard shortcuts, navigation techniques, and terminal-specific accessibility before diving into any CLI pathway.

Command-Line Foundation (Choose Your Path)

Students master terminal/command-line fundamentals before learning 3D design. You choose between three equivalent pathways based on your operating system and learning preferences.

Compare All Three Pathways

-> Command Line Interface Selection Guide

This guide provides a comprehensive comparison including:

- Feature matrix comparing all three shells
- Command comparison table (navigation, file operations, scripting)
- Learner profiles to help you choose
- Goal-based recommendations
- FAQ addressing common questions

Part 2A: PowerShell Foundation (Recommended for Windows Users)

Where to Start: PowerShell Curriculum Overview

Introductory & Reference Materials:

- PowerShell Introduction - Screen-reader-friendly quick-start guide (JAWS/NVDA)
- PowerShell Tutorial - Hands-on tutorial covering paths, navigation, and common operations

Lessons Included:

- PS-Pre: Your First Terminal - Opening PowerShell, first commands, screen reader tricks
- PS-0: Getting Started - Paths, shortcuts, tab completion
- PS-1: Navigation - Moving around the file system confidently
- PS-2: File and Folder Manipulation - Creating, editing, moving files
- PS-3: Input, Output, and Piping - Redirecting output, piping commands
- PS-4: Environment Variables and Aliases - Setting variables, creating shortcuts
- PS-5: Filling in the Gaps - Control flow, profiles, useful tricks
- PS-6: Advanced Terminal Techniques - Scripts, functions, professional workflows
- PowerShell Unit Test & Practice - Comprehensive self-assessment

Time commitment: 30-45 hours (for screen reader users) Best for: Users who want modern Windows features and advanced automation Skills: Terminal navigation, piping, advanced scripting, professional automation

Part 2B: Windows Command Prompt Foundation (Simplified Alternative)

Where to Start: CMD Curriculum Overview

Lessons Included:

- CMD-Pre: Your First Terminal - Opening CMD, first commands, screen reader tricks
- CMD-0: Getting Started - Paths, shortcuts, command basics
- CMD-1: Navigation - Moving around the file system confidently
- CMD-2: File and Folder Manipulation - Creating, editing, moving files
- CMD-3: Input, Output & Redirection - Redirecting output, piping commands
- CMD-4: Environment Variables & Shortcuts - Setting variables, shortcuts
- CMD-5: Filling in the Gaps - Batch files, advanced techniques
- CMD-6: Advanced Terminal Techniques - Scripts, automation, professional workflows
- CMD Unit Test & Practice - Comprehensive self-assessment

Time commitment: 30-45 hours (for screen reader users) Best for: Absolute beginners or users who prefer simplicity Skills: Terminal navigation, file operations, batch scripting, basic automation

Part 2C: Git Bash Foundation (Cross-Platform Skills)

Where to Start: Git Bash Curriculum Overview

Introductory & Reference Materials:

- Git Bash Introduction - Screen-reader-friendly quick-start guide (JAWS/NVDA)
- Git Bash Tutorial - Hands-on tutorial covering paths, navigation, and common operations
- Screen Reader Accessibility Guide for Git Bash - NVDA and JAWS configuration for Git Bash

Lessons Included:

- GitBash-Pre: Your First Terminal - Opening Git Bash, first commands, screen reader tricks
- GitBash-0: Getting Started - Paths, shortcuts, Unix path notation
- GitBash-1: Navigation - Moving around the file system confidently
- GitBash-2: File and Folder Manipulation - Creating, editing, moving files
- GitBash-3: Input, Output & Piping - Redirecting output, piping commands
- GitBash-4: Environment Variables & Aliases - Setting variables, creating shortcuts
- GitBash-5: Filling in the Gaps - Shell profiles, history, debugging
- GitBash-6: Advanced Terminal Techniques - Scripts, functions, professional workflows

- GitBash Unit Test & Practice - Comprehensive self-assessment

Time commitment: 20-25 hours (for screen reader users) Best for: Users who want cross-platform skills (Windows, macOS, Linux) Skills: Unix/bash commands, shell scripting, cross-platform automation

Important

Choose ONE pathway and complete it fully. All three teach identical fundamental concepts using different tools. Each integrates fully with 3D design workflows.

3dMake Foundation & Design (11 Lessons + 6 Appendices)

Where to Start: 3dMake Introduction

Students build parametric 3D designs using OpenSCAD and automate the workflow with 3DMake.

Main Curriculum: 11 Progressive Lessons

Part	Lessons	Focus	Duration
Foundations	1-3	Environment setup, primitives, parametric design + advanced language features	4-5 hours
Verification & Safety	4-5	AI verification, safety protocols, materials	2-3 hours
Applied Projects	6-8	Practical commands, text functions, transforms, assembly patterns	5-6 hours
Advanced Topics	9-10	Automation with file I/O, troubleshooting, mastery	3-4 hours
Leadership	11	Stakeholder-centric design	2-3 hours

Total: 45-55 hours instruction + practice projects

What's New (v2.1):

- Lesson 3: Added Advanced Programming Concepts (Conditionals, Type Testing, List Comprehensions, Special Variables, Functions)
- Lesson 6: Added String Functions for dynamic naming and labeling
- Lesson 7: Added Mathematical Functions (trigonometry, rounding, power/root, vector math)
- Lesson 8: Added Assembly Best Practices (snap-fits, threading, alignment, exploded views)
- Lesson 9: Added File Import/Export (import, include vs use, multi-file projects)

6 Comprehensive Reference Appendices

- Appendix A: Comprehensive Slicing Guide - Complete reference for PrusaSlicer, Bambu Studio, Cura, and OrcaSlicer configuration
- Appendix B: Material Properties & Selection Guide - Detailed material reference including shrinkage data, print settings, and properties
- Appendix C: Tolerance Testing & Quality Assurance Matrix - Comprehensive QA procedures and tolerance validation methods
- Appendix D: PowerShell Integration for SCAD Workflows - Batch processing, automation scripts, and advanced workflow integration
- Appendix E: Advanced OpenSCAD Concepts - Specialized topics including gears, batch processing, performance, print orientation, and recursion
- Appendix F: 3dMake Code Examples & Assets - OpenSCAD code examples and reference designs

The Accessible Toolchain

Screen Reader Compatibility Throughout

This course uses tools designed for screen reader access:

- Terminal/Command line - Text-based, fully accessible to NVDA, JAWS, VoiceOver
- OpenSCAD - Free, open-source text-based CAD (no visual-only GUI dependency)
- 3DMake - Command-line build tool eliminating GUI navigation
- Accessible editors - VS Code, Notepad++, Nano, Vim (all keyboard-driven, screen reader friendly)

See Screen Reader Coding Tips (NVDA & JAWS) for detailed keyboard shortcuts and configuration.

3DMake: Non-Visual Build Automation

3DMake makes the entire design-to-print pipeline accessible:

```
3dm build -> Compiles main.scad to main.stl
3dm info -> Validates geometry and runs diagnostics
3dm slice -> Prepares model for printing
```

- No GUI navigation needed
- Automation eliminates repetitive manual steps
- Configuration files store parameters as human-readable text
- Error reporting is text-based (screen reader accessible)
- Works with command-line slicers

Iterative, Non-Visual Design

Students learn to design through code and testing, not visual previews:

- Write parametric OpenSCAD code in accessible editors
- Run `3dm build` to compile to printable file
- Use measurement-based verification (calipers, scales, functional testing)
- Iterate by editing parameters and rebuilding
- No reliance on 3D preview or visual feedback

Project-Based Learning

Every 3dMake lesson includes hands-on projects:

- Lesson 6: Keycap with embossed text (3dm commands)
- Lesson 7: Phone stand (parametric transforms)
- Lesson 8: Stackable bins (interlocking features, tolerances)
- Lesson 9: Keychain automation (PowerShell batch processing)
- Lesson 10: QA testing + accessibility audit (measurement, troubleshooting)
- Lesson 11: Beaded jewelry holder (stakeholder-driven design)

Each project requires:

- Parametric OpenSCAD code (clean, well-commented)
- Functional prototype (tested, iterated)
- Complete documentation (reflections, measurements, decisions)

Learning Support

Reference Materials

Quick navigation to common topics:

OpenSCAD Cheat Sheet - Syntax quick-reference 3dMake Setup Guide - Installation walkthrough VSCode Setup Guide - Accessibility configuration Vocabulary Glossary - Course terminology Filament Comparison Table - Material reference Master Rubric - Project assessment criteria

Navigation

SUMMARY.md - Complete table of contents Curriculum Guide - Detailed overview of all lessons and appendices Quick Reference - At-a-glance command and syntax reference Appendices - 3dMake reference materials

Supplemental Resources & Textbooks

This course is enhanced by comprehensive textbooks and companion materials from the *Programming with OpenSCAD* project:

Textbook options (EPUB Format)

- Programming with OpenSCAD: A Beginner's Guide to Coding 3D-Printable Objects - Comprehensive reference covering OpenSCAD syntax, geometry concepts, and design patterns. Ideal for deep dives into specific topics and as a reference guide throughout the course.
- Simplifying 3D Printing with OpenSCAD - Focused on practical workflows, optimization, and real-world printing scenarios.

Companion Teaching Resources

Practice Worksheets - Printable worksheets for visualization practice, decomposition exercises, vocabulary building, and assessment. Visual Quick Reference - Command syntax guides and geometry reference. Code Solutions Repository - Working OpenSCAD examples organized by topic (3D shapes, transformations, loops, modules, if-statements, advanced techniques).

Getting Started

For Students:

1. Start with Setup & Accessibility Fundamentals
2. Read Command Line Interface Selection Guide to choose your CLI pathway
3. Complete your chosen CLI Foundation (PowerShell, CMD, or Git Bash)
4. Begin 3dMake Introduction
5. Follow Lesson 1: Environmental Configuration
6. Continue through Lesson 11

For Instructors:

1. Review Curriculum Guide
2. Use 11 Teacher Templates for assessment
3. Reference Master Rubric for grading
4. Check Syllabus for course policies and learning progression

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

3D Design & Printing Curriculum - Non-Visual Toolchain Edition

Author: Michael Ryan Hunsaker, M.Ed., Ph.D. Last Updated: 2026-02-22 Target Audience: Blind and visually impaired high school students; anyone learning 3D design and printing through screen reader-accessible workflows.

Overview

This curriculum teaches 3D design and digital fabrication using a fully accessible, command-line-driven toolchain centered on OpenSCAD (text-based CAD), 3DMake (non-visual build automation), and accessible editors (VS Code, Notepad++, command-line editors) with screen reader support. Students progress from foundational command-line skills through guided projects to real-world, stakeholder-driven design challenges.

Who This Course Is For

This course is explicitly designed for blind and visually impaired learners who use screen readers (NVDA, JAWS, VoiceOver). It eliminates GUI navigation and visual feedback in favor of keyboard-driven, text-based workflows that screen readers can fully access.

Accessibility is not an add-on. It is the foundation of every tool, workflow, and lesson in this curriculum.

Core Philosophy

1. Text-First Design: All core work happens in text editors and command-line interfaces - no graphical CAD previews, no mouse-dependent menu navigation.

2. Parametric Thinking: Students learn to express geometry as code using OpenSCAD, enabling precise, reproducible, and iterable designs without visual feedback.
3. Automation and Independence: 3DMake automates the journey from code to printed object, handling compilation, slicing orchestration, and metadata management through simple command-line commands and text configuration files.
4. Screen Reader Mastery: Students develop fluency with accessibility technologies (NVDA, JAWS, VoiceOver) and accessible editors, building skills that apply to careers in software, engineering, and digital fabrication.
5. Real-World Impact: Projects culminate in designing assistive-technology solutions for real stakeholders, combining technical skill with human-centered design and documentation.

Curriculum Structure & Scope/Sequence

Setup & Accessibility Fundamentals (Prerequisite - 2-3 hours)

Start here: Screen Reader Accessibility Guide

Before choosing a command-line pathway, students optimize their screen reader setup for terminal work.

Component	Duration	Content
Screen Reader Accessibility Guide	1-1.5 hours	NVDA/JAWS reference for PowerShell, CMD, and Git Bash; keyboard shortcuts
Screen Reader Choice: Windows CLI	30 min	Comparing NVDA, JAWS, Narrator, and Dolphin; choosing the right tool
Braille Display & Terminal Mode	30 min	Optional: configuring refreshable braille displays for terminal work
Editor Selection and Setup	30 min	Choosing Notepad, Notepad++, or VS Code; configuring indent announcement

Command-Line Foundation (Choose Your Path)

Start here: Command Line Interface Selection Guide

Students master terminal/command-line fundamentals before learning 3D design. Choose one of three equivalent pathways based on your operating system and preferences. All three pathways teach the same concepts and prepare you equally well for 3dMake work.

Important

Choose ONE pathway and complete it fully. All three teach identical fundamental concepts using different tools. Each integrates fully with 3D design workflows.

Pathway A: PowerShell Foundation (Recommended for Windows)

Total Duration: 30-45 hours Start here: PowerShell Curriculum Overview

Component	Duration	Content
PowerShell Introduction	20-30 min	Screen-reader-friendly quick-start (JAWS/NVDA); essential commands overview
PowerShell Tutorial	30-45 min	Hands-on tutorial: paths, navigation, wildcards, running scripts
PS-Pre: Your First Terminal	2-2.5 hours	Opening PowerShell, first commands, basic navigation, screen reader tricks
PS-0: Getting Started	1.5 hours	Paths, shortcuts, tab completion
PS-1: Navigation	2-2.5 hours	Moving around the file system confidently
PS-2: File & Folder Manipulation	2.5-3 hours	Creating, editing, moving, copying, deleting files and directories
PS-3: Input, Output & Piping	2.5-3 hours	Redirecting output, piping commands, understanding data flow

Component	Duration	Content
PS-4: Environment Variables & Aliases	2-2.5 hours	Setting variables, creating shortcuts, persistent configurations
PS-5: Filling in the Gaps	2.5-3 hours	Control flow, profiles, useful tricks, scripting fundamentals
PS-6: Advanced Terminal Techniques	4-4.5 hours	Scripts, functions, loops, professional workflows, automation patterns
PowerShell Unit Test & Practice	2-3 hours	Practice exercises, assessment, reinforcement

Outcomes: Terminal fluency, file system mastery, basic scripting, screen reader optimization, automation readiness

Pathway B: Windows Command Prompt (CMD) (Simpler alternative)

Total Duration: 30-45 hours Start here: CMD Curriculum Overview

Component	Duration	Content
CMD-Pre: Your First Terminal	2-2.5 hours	Opening CMD, first commands, basic navigation, screen reader tricks
CMD-0: Getting Started	1.5 hours	Paths, shortcuts, command basics
CMD-1: Navigation	2-2.5 hours	Moving around the file system confidently
CMD-2: File & Folder Manipulation	2.5-3 hours	Creating, editing, moving, copying, deleting files and directories

Component	Duration	Content
CMD-3: Input, Output & Redirection	2-2.5 hours	Redirecting output, piping commands, understanding data flow
CMD-4: Environment Variables & Shortcuts	2-2.5 hours	Setting variables, creating shortcuts, persistent configurations
CMD-5: Filling in the Gaps	2.5-3 hours	Batch files, advanced techniques, scripting fundamentals
CMD-6: Advanced Terminal Techniques	3-3.5 hours	Scripts, automation patterns, professional workflows
CMD Unit Test & Practice	2-3 hours	Practice exercises, assessment, reinforcement

Outcomes: Terminal fluency, file system mastery, batch scripting, screen reader optimization, automation readiness

Pathway C: Git Bash (Best for macOS/Linux or cross-platform development)

Total Duration: 20-25 hours Start here: Git Bash Curriculum Overview

Component	Duration	Content
Git Bash Introduction	20-30 min	Screen-reader-friendly quick-start (JAWS/NVDA); essential commands
Git Bash Tutorial	30-45 min	Hands-on tutorial: paths, navigation, wildcards, running scripts
Screen Reader Accessibility Guide for Git Bash	30-45 min	NVDA and JAWS configuration specific to Git Bash

Component	Duration	Content
GitBash-Pre: Your First Terminal	2-2.5 hours	Opening Git Bash, first commands, basic navigation, screen reader tricks
GitBash-0: Getting Started	1.5 hours	Unix-style paths, shortcuts, command basics, Windows path conversion
GitBash-1: Navigation	2-2.5 hours	Moving around the file system confidently
GitBash-2: File and Folder Manipulation	2-2.5 hours	Creating, editing, moving, copying, deleting files and directories
GitBash-3: Input, Output & Piping	2-2.5 hours	Redirecting output, piping with grep/sort/wc, understanding data flow
GitBash-4: Environment Variables & Aliases	1.5-2 hours	Setting variables, creating aliases, editing .bashrc
GitBash-5: Filling in the Gaps	2-2.5 hours	Shell profiles, command history, debugging
GitBash-6: Advanced Terminal Techniques	2.5-3.5 hours	Shell scripts, functions, loops, professional workflows
GitBash Unit Test & Practice	2-2.5 hours	Practice exercises, assessment, reinforcement

Outcomes: Terminal fluency, file system mastery, bash scripting, version control basics, automation readiness

Common Outcomes (All Pathways)

- Comfort with terminal/command-line interface
- File system navigation and manipulation

- Basic scripting and automation
- Screen reader optimization for terminal work
- Foundation for 3DMake automation tasks

3dMake Foundation (Main Curriculum - 16-20 hours)

Start here: 3dMake Introduction

11 progressive lessons building from foundational concepts to leadership-level design thinking, organized in 5 parts. Version 2.1 adds comprehensive advanced programming and design topics throughout.

Foundations (Lessons 1-3 | ~4-5 hours)

Lesson	Focus	Duration	Project
Lesson 1	Environmental Configuration + Code Documentation Standards	60-90 min	None
Lesson 2	Primitives & Boolean Operations + Modifier Characters Debugging	75-90 min	None
Lesson 3	Parametric Architecture + Advanced Programming Concepts	90-120 min	None

Verification & Safety (Lessons 4-5 | ~2 hours)

Lesson	Focus	Duration	Project
Lesson 4	AI-Enhanced Verification & Feedback	45-60 min	None
Lesson 5	Safety Protocols & Material Introduction	60-90 min	None

Applied Projects (Lessons 6-8 | ~5-6 hours)

Lesson	Focus	Duration	Project
Lesson 6	Practical 3dm Commands + String Functions	75-105 min	Customizable Keycap
Lesson 7	Parametric Transforms + Math Functions	90-120 min	Phone Stand
Lesson 8	Advanced Design + Assembly Best Practices	105-150 min	Stackable Bins

Advanced Topics (Lessons 9-10 | ~3-4 hours)

Lesson	Focus	Duration	Project
Lesson 9	Automation + File Import/Export (requires CLI Foundation)	75-105 min	Batch Processing Automation
Lesson 10	Troubleshooting & Mastery with Measurement	120-150 min	QA Testing + Accessibility Audit

Leadership (Lesson 11 | ~2 hours)

Lesson	Focus	Duration	Project
Lesson 11	Stakeholder-Centric Design & Design Thinking	90-120 min	Beaded Jewelry Holder

Total: 16-20 hours of instruction + projects

Reference Appendices

Located in `3dMake_Foundation/` alongside the lessons:

Appendix	Title	Use When
Appendix A	Comprehensive Slicing Guide	Slicing questions, switching slicers, quality issues
Appendix B	Material Properties & Selection Guide	Choosing material, troubleshooting prints, cost analysis
Appendix C	Tolerance Testing & Quality Assurance Matrix	Starting a project, verifying dimensions, quality issues
Appendix D	PowerShell Integration for SCAD Workflows	Automating tasks, testing variations, batch printing
Appendix E	Advanced OpenSCAD Concepts	Building mechanical systems, optimizing complex models
Appendix F	3dMake Code Examples & Assets	Reference designs, working code examples

Learning Progression: Student Roles

Students move through roles across the curriculum:

Phase	Role	Core Tools	Focus
CLI Foundation	Observer/Learner	Terminal, command line	CLI fundamentals and keyboard navigation
3dMake Lessons 1-5	Observer/Learner	OpenSCAD, 3DMake, editor	Using CLI tools, safety, concepts, measurement
3dMake Lessons 6-8	Operator	Editor, OpenSCAD, 3DMake, slicer	Hands-on practice with structured projects
3dMake Lessons 9-10	Designer	Full toolchain	Parametric design, automation, troubleshooting
3dMake Lesson 11	Problem-Solver	Full toolchain + documentation	Stakeholder design, real-world impact

The Accessible Toolchain: How It Works

OpenSCAD - Text-Based 3D Design

OpenSCAD is a free, open-source CAD tool that uses a programming language to describe 3D geometry. Students write code that defines shapes, transforms them, and combines them using Boolean operations.

Why OpenSCAD?

- Screen reader friendly: All work happens in a text editor; no visual-only 3D preview.
- Repeatable: Code is version-controlled, documented, and shareable.
- Parametric: Variables allow students to design once and generate variations by changing numbers.
- No visual dependency: Students reason about geometry through code structure and testing.

3DMake - The Non-Visual Build Bridge

3DMake is a command-line tool that automates the journey from OpenSCAD code to a printable file:

```
3dm build
3dm info
3dm slice
```

Why 3DMake?

- No GUI navigation: All interaction is keyboard-driven and text-based.
- Automation: Eliminates repetitive manual steps.
- Metadata tracking: Configuration files store parameters as human-readable text.
- Error reporting: Diagnostic output is text that screen readers can read aloud.

Accessible Editors

Students write OpenSCAD code using screen reader-accessible editors:

- VS Code (Windows, macOS, Linux): Industry-standard with built-in screen reader support
- Notepad++ (Windows): Lightweight, keyboard-driven, excellent screen reader support
- Command-line editors (Nano, Vim, Emacs): Full keyboard control, no mouse needed

Prerequisites by Section

Section	Prerequisites	What You'll Learn
Setup	None - start here	Screen reader optimization, editor selection, accessibility setup
CLI Foundation	Setup	Terminal basics, keyboard navigation, file operations, basic scripting
3dMake Lessons 1-5	Setup (CLI Foundation recommended)	3D printing concepts, safety, measurement, OpenSCAD basics, debugging
3dMake Lessons 6-8	Lessons 1-5	Building projects, parametric design, transforms, tolerances
3dMake Lessons 9-10	Lessons 6-8 + CLI Foundation required	Automation, troubleshooting, advanced measurement and QA
3dMake Lesson 11	Lessons 9-10	Stakeholder design, real-world prototyping, leadership

Grading Rubric

All projects are scored on a 0-9 scale across three equally weighted categories (3 points each):

Category	Points	What We Measure
Problem & Solution	0-3	Does the design solve the stated problem? Are all functional requirements met?
Design & Code Quality	0-3	Is the OpenSCAD code clean, well-commented, and well-structured? Does the print work well? Is there evidence of iteration?
Documen-tation	0-3	Are all sections complete? Are reflections thoughtful and specific? Are measurements recorded?

Category 1: Problem & Solution (0-3 points)

Score	Description
3	The prototype clearly and effectively solves the stated problem. All functional requirements are met. The solution shows evidence of testing against the requirements.
2	The prototype mostly meets the problem. Most functional requirements are met. Minor gaps between the design and the requirements.

Score	Description
1	The prototype partially addresses the problem. Several functional requirements are not met or were not clearly tested.
0	The prototype does not address the stated problem, or no functional requirements were established.

Category 2: Design & Code Quality (0-3 points)

OpenSCAD code is central to this course. We evaluate the clarity, structure, and documentation of your code as much as the print quality.

Score	Description
3	Code is clean, well-organized, and thoroughly commented. Variables/modules are used appropriately. Print quality is excellent. Design shows original thinking and at least one meaningful iteration.
2	Code works but lacks clear structure or comments. Variables are used but could be better named. Print quality is acceptable. Some iteration evident.
1	Code is functional but poorly organized. Comments are minimal or missing. Print quality has defects. Little or no iteration.
0	Code does not work, is not submitted, or shows no original thinking. Print is not functional.

Category 3: Documentation (0-3 points)

Score	Description
3	All required sections are present, complete, and specific. Reflections are thoughtful and reference specific decisions, problems encountered, and learning. Photos/measurements are included.
2	Most required sections are present. Some sections are vague or missing detail. Reflections show some thought but are brief or generic.
1	Documentation is incomplete. Major sections are missing or consist of one-line responses. Reflections are minimal.
0	Documentation is not submitted or is essentially empty.

Score Interpretation

Total Score	Interpretation	Next Step
8-9	Excellent work	Move on to next project

Total Score	Interpretation	Next Step
6-7	Good work with room for improvement	Move on; instructor may suggest revisiting one element
4-5	Meets basic expectations	Resubmission of specific weak areas recommended
2-3	Does not meet expectations	Resubmission required
0-1	Missing major deliverables	Meet with instructor; create a completion plan

Resubmission Policy

Students may resubmit any project as many times as they need to improve their score. Resubmissions must include a one-paragraph explanation of what was changed and why. The resubmission score replaces the original score.

Quick Links to Essential Tools & Setup

Core Design Toolchain

OpenSCAD

- OpenSCAD Download - Free, cross-platform CAD (all major OS)
- OpenSCAD Documentation - Official reference
- OpenSCAD Cheat Sheet - Quick syntax reference

3DMake

- 3DMake Documentation & Installation - Command-line build tool for OpenSCAD
- 3dMake Quick Reference - Command and workflow reference

Editors

- VS Code Download - Free, screen-reader-accessible code editor
- Notepad++ Download - Free, lightweight Windows editor
- Editor Selection and Setup Guide - Accessibility-focused setup guide

Screen Reader & Accessibility

Screen Readers

- NVDA Download - Free, open-source screen reader (Windows)
- JAWS Screen Reader - Commercial screen reader (Windows, macOS)

Accessibility Configuration

- Screen Reader Accessibility Guide - Comprehensive terminal accessibility guide
- Screen Reader Coding Tips (NVDA & JAWS) - Keyboard shortcuts and configuration
- VSCode Setup Guide - Accessibility-focused editor configuration
- Git Bash Screen Reader Guide - NVDA and JAWS configuration for Git Bash

Slicing Software

- PrusaSlicer
- Bambu Studio
- Cura
- OrcaSlicer
- Appendix A: Comprehensive Slicing Guide - Detailed setup guides for all major slicers

Supplemental Textbooks

- Programming with OpenSCAD: A Beginner's Guide
- Simplifying 3D Printing with OpenSCAD
- Programming with OpenSCAD Companion Resources
- Code Solutions Repository

Local Resources: Utah Makerspaces & Community Printing

Public Library Make Spaces

Salt Lake City Public Library

- SLC Public Creative Lab - Main Library (Level 1)
  - Hardware: Prusa i3 MK3, LulzBot Taz 5, Elegoo Mars 2 (resin)
  - Pricing: Free for prints under 6 hours; $0.50/hr + material cost otherwise

Salt Lake County Library System

- County Library ”Create” Spaces
  - Hardware: Flashforge Adventurer 5M Pro, LulzBot Workhorse, laser cutters
  - Pricing: $0.06 per gram of filament used

Makerspaces & Community Centers

Make Salt Lake

- Location: 663 W 100 S, Salt Lake City, UT 84101
- Website: https://makesaltlake.org/
- Equipment: Full metal shop, CNC machines, large-scale FDM and resin printing

University of Utah Maker Spaces

- Lassonde Studios
- Marriott Library ProtoSpace

Troubleshooting & Getting Help

If you're stuck:

1. Check Common Issues and Solutions
2. Check Diagnostic Checklist
3. Post in OpenSCAD Discord or Reddit
4. Visit your local makerspace for hands-on support

For accessibility support:

- Contact your NVDA/JAWS vendor directly for technical assistance
- Refer to the Screen Reader Accessibility Guide
- Check the Git Bash Screen Reader Guide if using Git Bash

Screen Reader Accessibility Guide for Command-Line Terminals

Target Users: NVDA, JAWS, Windows Narrator, and Dolphin SuperNova users using PowerShell, CMD, or Git Bash
Last Updated: February 2026

This guide is used throughout the Command-Line Foundation curriculum (PowerShell, CMD, and Git Bash pathways) to help screen reader users navigate and work efficiently with the terminal.

Table of Contents

Which Terminal Should I Use?

All three command-line shells (PowerShell, CMD, Git Bash) work well with screen readers. Choose the one that best fits your setup:

Feature	PowerShell	CMD	Git Bash
Platform	Windows only	Windows only	Windows, macOS, Linux
Screen Reader Support	Excellent	Excellent	Excellent
Learning Curve	Moderate	Gentle	Moderate-Steep
Use Case	Modern automation	Familiar simplicity	Cross-platform/Git workflows
Recommended For	3D design automation	Getting started easily	Software developers

Key Point: Choose one pathway and complete it fully. All three teach the same skills and prepare you equally well for 3dMake work. Don't switch mid-curriculum.

Getting Started with Screen Readers

Which Screen Reader Should I Use?

NVDA, JAWS, Windows Narrator, and Dolphin SuperNova all work with the three shells covered in this curriculum. Here is a brief comparison:

Feature	NVDA	JAWS	Windows Narrator	Dolphin SuperNova
Cost	Free (open-source)	Commercial (paid)	Free (built into Windows)	Commercial (paid)
Installation	Simple download	Complex but thorough	Already installed	Standard installer
Terminal Support	Excellent (all shells)	Excellent (all shells)	Good (all shells)	Good (all shells)
Learning Curve	Gentle	Steeper	Minimal	Moderate
Customization	Good (add-ons)	Extensive (scripts)	Limited	Good

Recommendation: If you are new to screen readers, start with NVDA (free, well-documented, good community support) or Windows Narrator (already on your computer, no installation needed). JAWS and Dolphin SuperNova are excellent choices if you already own a license or your organization provides one.

Before You Start

1. Make sure your screen reader is running **before** opening your terminal.
2. Open your terminal (PowerShell, CMD, or Git Bash) and let your screen reader read the title and prompt.
3. If you do not hear anything, press **Alt+Tab** to cycle windows and find your terminal.
4. Use your screen reader's review/virtual cursor to understand the layout.

NVDA-Specific Tips

NVDA is free and available from https://example.com

These tips work across all shells: PowerShell, CMD, and Git Bash.

Key Commands for Terminals

Command	What It Does
NVDA+Home	Read the current line (your command or output)
NVDA+Down Arrow	Read from cursor position to end of screen
NVDA+Up Arrow	Read from top of screen to cursor
NVDA+Page Down	Read next page
NVDA+Page Up	Read previous page
NVDA+F7	Open the Elements List / Review Mode viewer
NVDA+Shift+Right Arrow	Read next word
NVDA+Shift+Down Arrow	Read entire visible screen
NVDA+End	Jump to end of line
NVDA+Home	Jump to start of line
NVDA+Numpad 5	Announce the character under the review cursor
NVDA+Shift+Numpad 5	Announce the word under the review cursor

> **Note on NVDA key:** On desktop keyboards the NVDA modifier is typically the **Insert** key. On laptops without a numpad the NVDA key is often **CapsLock**. You can change this in NVDA Preferences □ Keyboard.

Example: Reading Long Output

Scenario: You ran a command and it listed 50 files. You cannot hear them all at once.

Solution with NVDA:

1. After the command finishes, press **NVDA+Home** to read the current line.
2. Press **NVDA+Down Arrow** repeatedly to read all output line by line.
3. Or redirect to a file first: `command > output.txt`, then open with `notepad output.txt` for more comfortable reading.

NVDA Settings for Terminals

1. Press **NVDA+N** to open the NVDA menu.
2. Go to **Preferences** ▯ **Settings** ▯ **Speech**.
3. Increase or decrease verbosity to taste.
4. Under **Document Formatting**, enable **"Report indentation"** (important for reading code).
5. Under **Terminal**, enable **"Speak typed characters"** so you hear each key as you type.

JAWS-Specific Tips

JAWS is a commercial screen reader available from https://example.com

These tips work across all shells: PowerShell, CMD, and Git Bash.

Key Commands for Terminals

Command	What It Does
Insert+Down Arrow	Read all / say all from cursor downward
Insert+Up Arrow	Re-read current line
Insert+Page Down	Read next page of terminal output
Insert+Page Up	Read previous page of terminal output
Insert+End	Jump to end of text on screen
Insert+Home	Jump to start of text on screen
Insert+Ctrl+Down	Read to end of screen
Insert+Ctrl+Up	Read to beginning of screen
Insert+Shift+Page Down	Select and read next page
Insert+F3	Open JAWS menu
Insert+Z	Toggle JAWS virtual cursor on/off
Insert+F2	Open JAWS Manager (Settings Center)

Note on JAWS key: The JAWS modifier is typically **Insert** on a standard keyboard. On laptops, JAWS can be configured to use **CapsLock** as the modifier instead.

Example: Reading Long Output

Scenario: You ran a command and saved output to a file.

Solution with JAWS:

1. Open the file in Notepad: `notepad file.txt`
2. In Notepad, press **Insert+Ctrl+Down** to hear all content from the top.
3. Use **Insert+Down Arrow** to read line by line at your own pace.
4. Use **Insert+F** (JAWS Find) to search for specific text within the file.

JAWS Settings for Terminals

1. Press **Insert+F3** to open the JAWS menu.
2. Go to **Options** ▯ **Settings Center** (or **Utilities** ▯ **Settings Manager** in older versions).
3. Search for **”terminal”** or **”console”**.
4. Enable **”Announce output”** and verify **”Speak when program speaks”** is on.
5. For indent reporting, search for **”Indent”** and set to **”Tones”** or **”Tones and Speech”**.

Windows Narrator-Specific Tips

Windows Narrator is built into Windows 10 and Windows 11 at no extra cost. Enable it via **Settings** ▯ **Accessibility** ▯ **Narrator** or press **Windows+Ctrl+Enter**.

These tips work across all shells: PowerShell, CMD, and Git Bash.

Key Commands for Terminals

Command	What It Does
Narrator+D	Read current line
Narrator+M	Read next line
Narrator+I	Read current item or focused element
Narrator+R	Read from cursor / read all from here
Narrator+Ctrl+R	Read page / read all
Narrator+Left/Right Arrow	Move to previous/next item
Narrator+Up/Down Arrow	Move to previous/next line in scan mode
Narrator+Enter	Activate/interact with current item
Narrator+Space	Toggle Narrator scan mode (browse vs. type mode)
Narrator+F1	Open Narrator help
Windows+Ctrl+Enter	Start or stop Narrator

Command	What It Does

> **Note on Narrator key:** The Narrator modifier is **CapsLock** or **Insert** by default. You can change this in **Settings □ Accessibility □ Narrator □ Keyboard shortcuts**.

Working with Terminals Using Narrator

Narrator works well in **focus mode** (the default when a terminal is active). In focus mode, arrow keys move through command history and output rather than triggering Narrator navigation.

Tips:

- After running a command, press **Narrator+D** to read the current line.
- Use **Narrator+R** to read all output from the current position.
- For long outputs, always redirect to a file: `command > output.txt`, then open in Notepad where Narrator's reading experience is more comfortable.
- In Windows Terminal, enable **"Accessible terminal"** mode in Windows Terminal settings for better Narrator integration.

Narrator Settings for Terminals

1. Open **Settings □ Accessibility □ Narrator**.
2. Under **"Change what you hear when typing"**, enable **"Hear characters as you type"**.
3. Under **"Change what Narrator reads"**, enable **"Read hints for controls and buttons"**.
4. Set verbosity level to **3 or 4** for a good balance of detail.
5. Under **"Choose a Narrator voice"**, select a voice you find comfortable for extended use.

Known limitation: Narrator has fewer customization options than JAWS or NVDA for advanced terminal work. If you find Narrator insufficient for complex workflows, consider switching to NVDA (free) or JAWS.

Dolphin SuperNova-Specific Tips

Dolphin SuperNova is a commercial screen reader (with optional magnification) available from https://example.com

These tips work across all shells: PowerShell, CMD, and Git Bash.

Key Commands for Terminals

Command	What It Does
Caps Lock+Down Arrow (or NumPad 2)	Read current line
Caps Lock+Numpad 5	Read current word
Caps Lock+Numpad 6	Read next word
Caps Lock+Numpad 4	Read previous word
Caps Lock+Numpad 8	Read from cursor to top
Caps Lock+Numpad 2	Read from cursor to bottom
Caps Lock+Numpad Plus	Say all / read all from current position
Caps Lock+L	Re-read current line
Caps Lock+Right Arrow	Read next character
Caps Lock+Left Arrow	Read previous character
Caps Lock+Page Down	Jump to end of document or screen
Caps Lock+Page Up	Jump to start of document or screen

Note on Dolphin key: Dolphin SuperNova typically uses **Caps Lock** as its modifier key (referred to as the "Dolphin key"). This can be changed in **SuperNova Control Panel □ Keyboard**.

Working with Terminals Using Dolphin SuperNova

1. Open SuperNova **before** opening your terminal application.
2. Dolphin SuperNova automatically tracks focus as you move between the editor, terminal, and file explorer.
3. In the terminal, use the **Dolphin key+Numpad Plus** (say all) to have the screen read after running a command.
4. For complex output, redirect to a file and open in Notepad: `command > output.txt`, then `notepad output.txt`.

Dolphin SuperNova Settings for Terminals

1. Open **SuperNova Control Panel** (press Caps Lock+SpaceBar, or click the SuperNova icon).
2. Go to **Speech □ Verbosity** and set a comfortable level (3 or 4 recommended for terminal work).
3. Go to **Speech □ Text Processing □ Indentation** and enable indentation announcement (important for code).
4. Go to **Braille □ Settings** if you are using a braille display alongside speech.
5. In **General □ Application Settings**, you can set terminal-specific verbosity so SuperNova behaves differently in your terminal versus other programs.

Note: Dolphin SuperNova's magnification features can be helpful if you have some remaining vision. Terminal text can be magnified while still receiving speech output.

General Terminal Accessibility

Understanding the Terminal Layout

All terminals (PowerShell, CMD, Git Bash) follow the same basic layout:

1. **Title bar:** Window name (e.g., ”Windows PowerShell”, ”Command Prompt”, ”Git Bash”)
2. **Content area:** Command history and output
3. **Prompt:** The area where you type (e.g., `PS>`, `C:\>`, `$`)

Your screen reader reads from top to bottom, but focus is at the prompt (bottom).

Navigation Sequence

When you open any terminal:

1. Your screen reader announces the window title.
2. Then it announces the prompt line.
3. Anything before the prompt is previous output.
4. Anything after the prompt is where new output will appear.

Reading Output Effectively

Strategy 1: Immediate Output (Small Amount)

- Run a command.
- Your screen reader announces output immediately.
- This works well for short outputs (a few lines).

Strategy 2: Large Output (Many Lines)

- Redirect to a file: `command > output.txt`
- Open the file: `notepad output.txt` (works in all shells on Windows)
- Read in Notepad — easier and more controllable for all screen readers.

Strategy 3: Filtering Output

- Use filtering commands to reduce output.
- PowerShell: `command | Select-String "pattern"`
- CMD: `command | find "pattern"`
- Git Bash: `command | grep "pattern"`

Working with Long Output

This is one of the most common challenges for screen reader users. Here are proven solutions:

Solution 1: Redirect to a File (Recommended)

```
command > output.txt
notepad output.txt
```

Advantages: Easy to navigate at your own pace, works with all screen readers, output does not scroll away, you can save it for later reference.

Solution 2: Use Pagination

```
command | more
```

Press **Space** to see the next page, **Q** to quit. Note: some screen readers struggle with `more`; Solution 1 is generally preferred.

Solution 3: Filter Output

```
# PowerShell
ls | Select-String "\.scad"

# CMD
dir | find ".scad"

# Git Bash
ls -la | grep ".scad"
```

Solution 4: Count Before Displaying

```
# PowerShell
(ls).Count
```

This tells you how many items there are before deciding whether to redirect to a file.

Keyboard Shortcuts Reference

Shell Navigation (Works Regardless of Screen Reader)

Key	Action	Why It Matters
Up Arrow	Show previous command	Repeat commands without retyping
Down Arrow	Show next command	Navigate through history
Tab	Auto-complete path or command name	Faster and more accurate
Shift+Tab	Cycle backward through completions	If Tab went too far
Home	Jump to start of command line	Edit beginning of a command
End	Jump to end of command line	Edit end of a command
Ctrl+A	Select all text on line	Copy entire command
Ctrl+C	Stop / interrupt running command	Abort long-running tasks
Ctrl+L	Clear screen	Start fresh visually
Enter	Run command	Execute what you typed

Screen Reader Navigation Quick Reference

Task	NVDA	JAWS	Windows Narrator	Dolphin SuperNova
Read current line	NVDA+Home	Insert+Up Arrow	Narrator+D	CapsLock+L
Read all from here	NVDA+Down Arrow	Insert+Ctrl+Down	Narrator+R	CapsLock+Numpad Plus
Read next line	Down Arrow (review)	Insert+Down Arrow	Narrator+M	CapsLock+Numpad 2
Read previous line	Up Arrow (review)	Insert+Up Arrow	Narrator+Up Arrow	CapsLock+Numpad 8
Jump to end of output	NVDA+End	Insert+End	Narrator+End	CapsLock+Page Down
Jump to start of output	NVDA+Home	Insert+Home	Narrator+Home	CapsLock+Page Up
Stop reading	Ctrl	Ctrl	Ctrl	Ctrl

Shell-Specific Differences

While all three shells work equally well with screen readers, there are differences in commands and syntax:

File Listing and Navigation

Task	PowerShell	CMD	Git Bash
List files	`ls` or `Get-ChildItem`	`dir`	`ls -la`
List names only	`ls -Name`	`dir /B`	`ls -1`
Change directory	`cd path`	`cd path`	`cd path`
Current location	`pwd`	`cd` (no args)	`pwd`
Create folder	`mkdir foldername`	`mkdir foldername`	`mkdir foldername`
Delete file	`Remove-Item file.txt`	`del file.txt`	`rm file.txt`
Home directory	`$HOME` or `~`	`%USERPROFILE%`	`~`

Output Redirection

Task	PowerShell	CMD	Git Bash
Save to file	`command > file.txt`	`command > file.txt`	`command > file.txt`
Append to file	`command >> file.txt`	`command >> file.txt`	`command >> file.txt`
Filter output	`command | Select-String "text"`	`command | find "text"`	`command | grep "text"`
Open file	`notepad file.txt`	`notepad file.txt`	`notepad file.txt` (Windows)

Scripting and Automation

Feature	PowerShell	CMD	Git Bash
File extension	`.ps1`	`.bat`	`.sh`
Comment	`#`	`REM` or `::`	`#`
Variable	`$var = "value"`	`set var=value`	`var=value`

Feature	PowerShell	CMD	Git Bash
Echo text	`Write-Host "text"`	`echo text`	`echo text`
Get help	`Get-Help command`	`help command`	`man command`

Choose ONE pathway and stick with it. Each curriculum teaches you the concepts using that specific shell's syntax. The accessibility experience is virtually identical across all four screen readers in all three shells — only the command syntax differs.

Troubleshooting

Problem 1: "I Can't Hear the Output After Running a Command"

Causes and Solutions:

1. **Cursor is not at the prompt** — Press **End** or **Ctrl+End** to go to the end of text, then use screen reader commands to review.
2. **Output scrolled off-screen** — Redirect to file: `command > output.txt`, then `notepad output.txt`.
3. **Screen reader focus is on window title, not content** — Press **Tab** or arrow keys to move into the content area.
4. **Large output overwhelming screen reader** — Use filtering: `command | grep "pattern"` (Git Bash), `command | find "text"` (CMD), or `command | Select-String "text"` (PowerShell).

Problem 2: "Tab Completion Isn't Working"

1. You must type at least one character before pressing Tab.
2. The folder or file must exist — run `dir` (CMD), `ls` (PowerShell/Git Bash) to check.
3. If multiple matches exist, press Tab again to cycle through them.

Problem 3: "Access Denied or Permission Denied"

1. Close your terminal, right-click it, and choose **Run as administrator**. Confirm the UAC prompt.
2. If the file is in use by another program, close that program and try again.
3. If the path contains spaces, use quotes: `cd "Program Files"`.

Problem 4: ”A Command Runs Forever and Won't Stop”

Press **Ctrl+C** to interrupt any running command.

Problem 5: ”I Need to Edit My Last Command”

1. Press **Up Arrow** to recall the previous command.
2. Use **Left/Right Arrow** keys to move through it.
3. Edit as needed and press **Enter** to run the modified version.

Problem 6: Screen Reader is Not Announcing Indentation

Proper indent announcement is critical for reading code. Enable it for your screen reader:

- **NVDA:** NVDA Menu Preferences Settings Document Formatting enable **”Report indentation”**, set to **”Tones and speech”**.
- **JAWS:** Settings Center search **”Indent”** enable **”Announce Indentation”**, set mode to **”Tones”** or **”Tones and Speech”**.
- **Windows Narrator:** Settings Accessibility Narrator Advanced Options enable **”Report indentation”** (limited options compared to NVDA/JAWS).
- **Dolphin SuperNova:** SuperNova Control Panel Speech Text Processing Indentation enable and set preferred announcement style.

Pro Tips for Efficiency

Create Aliases for Frequently Used Commands

PowerShell: `Set-Alias -Name ll -Value "Get-ChildItem -Name"`
CMD: Create a `.bat` file in a folder on your PATH.
Git Bash: `alias ll='ls -la'` (add to `~/.bashrc` to make permanent)

Use Command History

All shells support **Up/Down Arrow** to navigate history. In PowerShell, `history` shows a numbered list and `Invoke-History 5` reruns command number 5. In Git Bash, `history` and `!5` work similarly.

Redirect Everything to Files for Accessibility

If a command produces output, save it:

```
command > results.txt
notepad results.txt
```

This is always more accessible than reading from the terminal directly.

Quick Reference Card

```
EVERY SESSION STARTS WITH:
1. pwd / cd (where am I?)
2. dir / ls (what is here?)
3. cd path (go there)

LONG OUTPUT?
-> command > file.txt
-> notepad file.txt

STUCK?
-> Ctrl+C

WANT TO REPEAT?
-> Up Arrow

NEED HELP?
-> PowerShell: `Get-Help command-name`
-> CMD: `help command-name`
-> Git Bash: `man command-name`
```

Additional Resources

- **NVDA:** https://example.com
- **JAWS:** https://example.com
- **Windows Narrator:** https://example.com
- **Dolphin SuperNova:** https://example.com
- **PowerShell Docs:** https://example.com
- **CMD Documentation:** https://example.com
- **Git Bash / Git for Windows:** https://example.com
- **Accessibility Best Practices:** https://example.com

Choosing a Screen Reader for Windows Command-Line Work

Screen readers provide speech output and, where supported, braille output that make text-based interfaces accessible to people who are blind or have low vision. In command-line environments they convert terminal text, prompts, and keyboard navigation into audible or tactile feedback so users can work independently in shells such as Windows Terminal, Command Prompt, PowerShell, and Git Bash.

Choosing a screen reader is a personal decision. There is no single reader that is universally better or worse for every user. Preferences depend on workflow, training, budget, and comfort with customization. Each product has genuine strengths and trade-offs. If possible, try the available options and pick what feels most natural for you.

The Four Screen Readers Covered in This Curriculum

NVDA (NonVisual Desktop Access)

NVDA is a free, open-source screen reader developed by NV Access. It is one of the most widely used screen readers in the world.

Advantages:

- Free to download and use; supported by donations.
- Actively developed with frequent updates.
- Excellent support for Windows terminals (Command Prompt, PowerShell, Windows Terminal, Git Bash).
- Extensible through a rich add-on ecosystem.
- Strong community support, forums, and troubleshooting resources.
- Good braille display support via built-in braille drivers and the BRLTTY/LibLouis library.

Disadvantages:

- Advanced paid support is community-driven rather than available from a commercial vendor helpdesk.
- Some edge-case compatibility differences compared to commercial readers.

Cost: Free (donations welcomed at https://example.com).

Trial: Not applicable — NVDA is always free.

Download: https://example.com

JAWS (Job Access With Speech)

JAWS is a commercial screen reader developed by Freedom Scientific (part of Vispero). It is one of the most feature-rich and widely deployed screen readers in enterprise and educational settings.

Advantages:

- Mature product with decades of development and broad application compatibility.
- Extensive scripting system allows deep customization for specific applications.

- Vendor-provided paid support, training, and documentation.
- Advanced braille display support and configuration.
- Widely used in workplaces and schools, so shared knowledge base is large.

Disadvantages:

- Commercial product with significant licensing cost.
- JAWS demo mode limits session length to approximately 40 minutes before requiring a computer restart or session reset to continue. This can be disruptive during extended practice sessions.
- Some advanced customization (JAWS scripts) requires a learning curve.

Cost: Commercial. Typical U.S. pricing is several hundred dollars per year for a software maintenance agreement (SMA), or a higher one-time perpetual license fee. Check https://example.com for current pricing.

Trial: JAWS offers a time-limited demonstration mode (approximately 40 minutes per session). After the demo period expires, you must restart the computer to continue using JAWS without a license. This applies to the full JAWS installer — there is no separate shorter trial download.

Download: https://example.com

Windows Narrator

Windows Narrator is Microsoft's built-in screen reader, included with Windows 10 and Windows 11 at no extra cost. It has improved significantly with recent Windows releases.

Advantages:

- Already installed on every Windows 10 and Windows 11 computer — no download or installation needed.
- No cost.
- Simple to start: press **Windows+Ctrl+Enter** to toggle Narrator on and off.
- Reasonable support for Windows Terminal, Command Prompt, and PowerShell.
- Integrates tightly with Windows accessibility features.

Disadvantages:

- Fewer customization options than JAWS or NVDA for complex command-line workflows.
- Braille display support exists but is more limited than dedicated screen readers.
- Less powerful for advanced scripting, code editing, or rapid navigation in large terminal outputs.
- Less community documentation for terminal-specific workflows compared to NVDA and JAWS.

Cost: Free (included with Windows).

Trial: Not applicable — Narrator is always available on Windows.

Enable: Settings ▯ Accessibility ▯ Narrator, or press **Windows+Ctrl+Enter**.

Dolphin SuperNova

Dolphin SuperNova (sometimes referred to as "Dolphin Screen Reader" or simply "SuperNova") is a commercial product from Dolphin Computer Access. It is available in several editions: Magnifier, Magnifier & Speech, and Screen Reader. The Screen Reader and Magnifier & Speech editions provide full screen reader functionality.

Advantages:

- Commercial product with vendor support, training options, and documentation.
- Offers combined magnification and speech in a single product — useful for users with low vision who benefit from both.
- Good braille display support.
- Works with Windows terminals (Command Prompt, PowerShell, Windows Terminal).
- Full-featured 30-day trial license available for evaluation before purchase.

Disadvantages:

- Commercial licensing cost.
- Less widely used than NVDA or JAWS, so community resources and third-party documentation are more limited.
- Configuration for some terminal workflows may require additional setup.

Cost: Commercial. Pricing varies by edition (Magnifier only, Magnifier & Speech, or Screen Reader). Check `https://example.com` for current pricing. Expect costs broadly similar to other commercial screen readers.

Trial: Dolphin provides a full-featured 30-day trial license. This is more generous than JAWS's per-session demo and allows uninterrupted evaluation over a full month.

Download: https://example.com

Choosing Between Them for Command-Line Work

Use this decision guide to narrow your choice:

If budget is the primary constraint:
Start with NVDA or Windows Narrator. Both are free. NVDA is generally more

capable and better documented for terminal work. Windows Narrator requires no installation and is a good quick-start option.

If you already own or have access to JAWS or Dolphin through school or work:
Use what you have. Both work well for this curriculum.

If you have low vision (partial sight) in addition to using a screen reader:
Consider Dolphin SuperNova, which combines magnification and speech in one product, or use Windows Magnifier alongside NVDA.

If your organization requires vendor-supported software:
JAWS and Dolphin SuperNova both offer commercial support contracts.

If you are unsure:
Install NVDA (free, no risk) and try a few lessons. You can always switch later, and the terminal commands you learn work identically regardless of which screen reader you use.

Terminal Command Reference by Screen Reader

All four screen readers can perform the same tasks in the terminal. The table below shows the key actions and how to perform them with each reader. Exact key names can vary by keyboard layout and screen reader configuration.

Task	NVDA	JAWS	Windows Narrator	Dolphin SuperNova
Read current line or prompt	NVDA+Home (reads current line)	Insert+Up Arrow (re-reads current line)	Narrator+D (reads current line)	CapsLock+L (reads current line)
Read all output from here	NVDA+Down Arrow (read to end of screen)	Insert+Ctrl+Down (read to end of screen)	Narrator+R (read from cursor)	CapsLock+Numpad Plus (say all)
Move to previous line of output	Up Arrow in review mode	Insert+Up Arrow (line by line)	Narrator+Up Arrow (scan mode)	CapsLock+Numpad 8
Move to next line of output	Down Arrow in review mode	Insert+Down Arrow (line by line)	Narrator+Down Arrow (scan mode)	CapsLock+Numpad 2
Jump to end of screen/buffer	NVDA+End	Insert+End	Narrator+End	CapsLock+Page Down

Task	NVDA	JAWS	Windows Narrator	Dolphin SuperNova
Jump to start of screen/buffer	NVDA+Home	Insert+Home	Narrator+Home	CapsLock+Page Up
Recall last command (shell history)	Up Arrow (shell native; SR reads it)	Up Arrow (shell native; SR reads it)	Up Arrow (shell native; SR reads it)	Up Arrow (shell native; SR reads it)
Stop reading	Ctrl	Ctrl	Ctrl	Ctrl
Open screen reader settings	NVDA+N Preferences	Insert+F3 Options/Settings	Windows+Ctrl+N (Narrator settings)	CapsLock+SpaceBar (SuperNova control panel)

Notes on modifier keys:

- **NVDA key** is Insert by default; can be changed to CapsLock in NVDA Preferences Keyboard.
- **JAWS key** is Insert by default; can be changed to CapsLock in JAWS Settings Center.
- **Narrator key** is CapsLock or Insert; configurable in Settings Accessibility Narrator Keyboard shortcuts.
- **Dolphin key** is CapsLock by default; configurable in SuperNova Control Panel Keyboard.

Important Notes for This Curriculum

1. **All four screen readers work with all three shells** (CMD, PowerShell, Git Bash) covered in this curriculum. You do not need to change screen readers based on which shell pathway you choose.

2. **The best practice for all four screen readers** when dealing with long terminal output is to redirect it to a file and open it in Notepad:

```
command > output.txt
notepad output.txt
```

   This works in CMD, PowerShell, and Git Bash, and gives you a stable document that all four screen readers can read comfortably.

3. **Braille display users** can use a braille display alongside any of these four screen readers. See the Braille Display Setup guide for detailed instructions.

4. **The screen reader tips in each lesson** of this curriculum are written for NVDA, JAWS, Windows Narrator, and Dolphin SuperNova. If the specific key commands differ from what you hear when practicing, check your screen reader's documentation for the equivalent command — the underlying concept is always the same.

Additional Resources

Screen Reader	Official Documentation	Support / Help
NVDA	https://example.com	https://example.com
JAWS	https://example.com	https://example.com
Windows Narrator	https://example.com	https://example.com
Dolphin SuperNova	https://example.com	https://example.com

Using a Braille Display with the Curriculum

Many blind and low-vision programmers prefer tactile output alongside or instead of speech. A refreshable braille display presents terminal text as a row (or multiple rows) of braille cells and allows you to read at your own pace, reread lines without interrupting speech, and work quietly. This page covers how to connect and configure a braille display with each screen reader used in this curriculum.

How a Braille Display Works with a Terminal

A refreshable braille display uses small pins that rise and fall to form braille characters. When connected to a computer and paired with a screen reader, the display shows the line of text where the screen reader's focus or review cursor is located. You can pan left and right across a long line or press routing buttons to move the screen reader cursor to a specific position.

In a terminal environment:

- The display shows the current command line (what you are typing).
- After running a command, you can use your screen reader's review cursor and the display's pan keys to read through the output line by line.
- Routing keys (small buttons above or below each braille cell) let you move the cursor directly to that point in the text — useful for re-running or editing commands from history.

Connecting a Braille Display

USB Connection

1. Plug the display's USB cable into your computer.
2. Windows will attempt to install drivers automatically. If the display is not recognized, install the vendor's driver software first (see vendor links at the end of this page).
3. Open your screen reader's braille settings and select the connected display.

Bluetooth Connection

1. Put the braille display into pairing mode (refer to the device's quick-start guide).
2. On Windows, go to **Settings** ▯ **Bluetooth & devices** ▯ **Add a device** and pair the display.
3. Once paired, open your screen reader's braille settings and select the display.

Before Opening the Terminal

Always confirm the display is connected and showing output before opening your terminal. A quick test: open Notepad, type a few words, and verify they appear on the display. This confirms the screen reader is routing output correctly before you add the complexity of a terminal session.

Configuring Each Screen Reader for Braille

NVDA

NVDA includes built-in braille support for a wide range of displays without requiring separate vendor software for many devices.

1. Connect your display via USB or Bluetooth.
2. Open the NVDA menu: press **NVDA+N** (Insert+N or CapsLock+N depending on your modifier key setting).
3. Go to **Preferences** ▯ **Settings** ▯ **Braille** (or press **NVDA+Ctrl+B** as a shortcut on some versions).
4. In the **”Braille display”** dropdown, select your display from the list. NVDA supports automatic detection for many displays — you can also select **”Automatic”** and let NVDA detect it.

5. Set your preferred **braille output table** (e.g., Unified English Braille Grade 1 or Grade 2, or your country's national table).
6. Set **”Braille input table”** if you want to type commands using the display's braille keyboard (if equipped).
7. Enable **”Show cursor”** so you can see the caret position on the display.
8. Click **OK** and test by moving the cursor in Notepad.

Recommended NVDA braille settings for terminal work:

- Output table: Unified English Braille Grade 1 (uncontracted) is recommended for reading code — contractions in Grade 2 can make code harder to read.
- Cursor shape: Dots 7 and 8 (underline) is a common choice.
- Enable **”Word wrap”** to avoid cutting words at the display's edge.

NVDA braille keyboard shortcut reference (while braille display is active):

Action	NVDA Command
Pan braille display right	Display's right pan key
Pan braille display left	Display's left pan key
Toggle braille tethered to focus/review	NVDA+Ctrl+T
Move to previous braille line	NVDA+Shift+Up (or display key)
Move to next braille line	NVDA+Shift+Down (or display key)
Open braille settings	NVDA+Ctrl+B (some versions)

JAWS

JAWS supports a wide range of braille displays through its built-in braille manager and additional vendor drivers.

1. Connect your display via USB or Bluetooth.
2. Install the vendor's display driver if required (check the vendor's website — some displays work without a driver in JAWS, others need one).
3. Open the JAWS menu: press **Insert+F3** (or your JAWS key+F3).
4. Go to **Options ▯ Basics ▯ Braille** or use the **JAWS Settings Center** and search for **”Braille”**.
5. In Braille Settings, select your display model from the list.
6. Choose your **braille translation table** (e.g., English Unified Grade 1 for uncontracted, Grade 2 for contracted).
7. Configure **cursor routing** so that pressing a routing button on the display moves the JAWS cursor to that position.
8. Test in Notepad before moving to a terminal.

JAWS braille settings for terminal work:

- Use **Grade 1 (uncontracted) braille** for reading code and terminal output. Grade 2 contractions can make commands and file paths difficult to parse.
- Enable **”Show active cursor”** in JAWS braille settings.
- In the JAWS Settings Center, look for **”Braille for Applications”** — you can set different braille behavior for specific applications (e.g., a terminal vs. a word processor).

Accessing braille settings:

- **Insert+F3** Options Basics Braille
- Or: **Insert+F2** (JAWS Manager) Settings Center search ”braille”

Windows Narrator

Windows Narrator supports braille output through the **Windows braille service** (Windows 10 version 1903 and later, and Windows 11).

1. Connect your display via USB or Bluetooth.
2. Install the vendor's display driver if required.
3. Open **Settings** **Accessibility** **Narrator** **Braille** (Windows 11) or **Settings** **Ease of Access** **Narrator** **Use Braille** (Windows 10).
4. Turn on **”Install braille”** (first-time setup may require downloading the braille translation component — approximately 70 MB).
5. Select your **display brand and model** from the list.
6. Choose a **braille table** (Grade 1 recommended for terminal work).
7. Test in Notepad before using in a terminal.

Important Narrator braille notes:

- Narrator's braille support uses the **BRLTTY** back-end on Windows, which supports most mainstream displays.
- If your display is not listed, check for a Windows Update or visit the display vendor's website for a Windows Narrator-compatible driver.
- Narrator's braille customization is less extensive than NVDA's or JAWS's. If you need more control, consider using NVDA (free) for braille-heavy work.
- The **braille display showing Narrator output** does not change the fact that Narrator's keyboard commands remain the same — your display shows what Narrator is currently reading.

Dolphin SuperNova

Dolphin SuperNova includes integrated braille support as part of its screen reader editions (SuperNova Magnifier & Speech and SuperNova Screen Reader).

1. Connect your display via USB or Bluetooth.
2. Install the vendor's driver if required.

3. Open the **SuperNova Control Panel**: press **CapsLock+SpaceBar** or click the SuperNova icon in the taskbar.
4. Go to **Braille ▯ Display** and select your display model from the list.
5. Go to **Braille ▯ Translation** and select your braille table (Grade 1 / uncontracted recommended for code).
6. Go to **Braille ▯ Settings** to configure cursor style, word wrap, and other options.
7. Click **Apply** and test in Notepad.

Dolphin SuperNova braille tips for terminal work:

- SuperNova supports a wide range of displays from Freedom Scientific (Focus), HumanWare (Brailliant, BrailleNote as terminal), HIMS, Optelec, and others.
- Use **Grade 1 (uncontracted)** braille for terminal and code work.
- SuperNova's **”Braille cursor tracking”** setting determines whether the braille display follows the text cursor or the screen review cursor — set this to **”Cursor”** when working in a terminal.
- Dolphin provides a **braille viewer** on screen (a visual representation of what is on the braille display) — useful when setting up or troubleshooting.

Using the Braille Display in a Terminal

Once your display is configured, here is how to work in a terminal:

1. **Reading the prompt:** The display shows your current prompt (e.g., `C:\Users\YourName>` or `PS C:\>`). Pan right if the prompt is longer than your display width.
2. **Reading output:** After running a command, use your screen reader's review cursor commands (see the Screen Reader Accessibility Guide) to move through the output. The display shows one line at a time and updates as you move.
3. **Redirecting long output:** For long terminal output, redirect to a file and open in Notepad:

   ```
   command > output.txt
   notepad output.txt
   ```

   In Notepad, you can pan through the braille display comfortably without the output scrolling away.
4. **Using routing keys:** Press the routing key above a word or character to move your cursor to that position. This is especially useful for editing commands recalled from shell history.

Braille Grade and Code

For all screen readers, use **Grade 1 (uncontracted) braille** when working in terminals and with code. Contracted braille (Grade 2) uses abbreviations designed for reading prose, and those contractions will make command names, file paths, and code syntax difficult or impossible to read correctly.

Example: In Grade 2, the letters "cd" might be represented as a contraction meaning something other than the Change Directory command. In Grade 1, `cd` is always shown as the letters c and d.

Notetakers and BrailleNote / BrailleSense Devices

BrailleNote Touch+ (HumanWare) and BrailleSense (HIMS) devices run Android as their primary operating system. **They cannot run `3dMake` or Windows-based terminal workflows natively.** These devices are not supported as standalone authoring environments for this curriculum.

Workaround — using them as braille terminals: Both device families can function as a braille display when connected to a Windows computer via USB or Bluetooth. In that configuration:

- The device acts as a standard refreshable braille display.
- Your Windows screen reader (NVDA, JAWS, Narrator, or SuperNova) drives the output.
- Commands are typed on the Windows keyboard; the BrailleNote/BrailleSense shows the output as braille.

To set this up, refer to your device's documentation for "Terminal mode" or "PC connection mode" and then follow the screen reader braille configuration steps above for your chosen screen reader.

Multiline Braille Displays

Standard refreshable braille displays show a single line of braille (typically 32, 40, or 80 cells). Multiline displays show several lines simultaneously, which can significantly improve the experience of reading code or terminal output because you can see context above and below the current line.

Examples

Monarch (APH / HumanWare partnership):
A multiline braille device developed by the American Printing House for the Blind (APH) in partnership with HumanWare. It presents multiple lines of braille and supports graphical braille output, making it useful for reviewing blocks of

code, diagrams, and structured text. Confirm current driver and screen reader compatibility before purchase.

DotPad (Dot Inc.):
A multiline braille and tactile graphics display. Its multiline capability allows programmers to scan several lines of code or terminal output at once. Useful for reviewing structured output without panning repeatedly. Check current Windows compatibility and screen reader support status.

Graphiti and Graphiti Plus (Orbit Research):
Multiline tactile displays from Orbit Research designed for reading text and tactile graphics. Their expanded line count helps with reading code structure and terminal output. Check current driver support for NVDA, JAWS, Narrator, and SuperNova.

Considerations for Multiline Displays

- **Driver and screen reader support:** Multiline displays are newer technology and support varies. Confirm that your chosen screen reader (NVDA, JAWS, Narrator, or SuperNova) has a driver for the specific device before purchase.
- **Size and portability:** Multiline displays are larger and heavier than single-line models. Consider your workspace and whether you need portability.
- **Cost:** Multiline displays are significantly more expensive than single-line displays. Evaluate whether the workflow benefit justifies the cost for your situation.
- **Learning curve:** Using a multiline display effectively — navigating across lines, understanding the spatial layout — requires some practice beyond what is needed for single-line displays.

Troubleshooting

Display shows nothing / no braille output:

- Verify the USB or Bluetooth connection and check that the display is powered on.
- In your screen reader's braille settings, confirm the correct display model is selected.
- Try disconnecting and reconnecting. Restart the screen reader if needed.
- Install or reinstall the vendor's display driver.

Display shows output but navigation is out of sync:

- Toggle your screen reader's braille mode off and on (usually in the braille settings).

- Restart the screen reader (not the whole computer) and reopen the terminal.
- Try a different terminal emulator (Windows Terminal, Command Prompt, PowerShell) to see if the issue is terminal-specific.

Display driver not found:

- Go to the vendor's website (links below) and download the latest driver for your operating system version.
- Some displays require firmware updates to work with newer Windows versions.

Routing keys not moving the cursor:

- In your screen reader's braille settings, confirm that cursor routing is enabled.
- In NVDA, this is under Preferences Settings Braille enable ”Move system cursor when routing review cursor”.
- In JAWS, check the routing settings in the JAWS braille configuration.

Vendor Resources

Vendor	Products	Support / Drivers
HumanWare	Brailliant, BrailleNote Touch+	https://example.com
Freedom Scientific	Focus Blue series	https://example.com
HIMS	BrailleSense, Braille EDGE, Smart Beetle	https://example.com
Optelec	Braille STAR, Easy Link	https://example.com
Orbit Research	Graphiti, Orbit 20/40	https://example.com
APH	Monarch	https://example.com
Dot Inc.	DotPad	https://example.com
Dolphin	GuideConnect, braille accessories	https://example.com

---

Connecting a Braille Display: Summary Table

Screen Reader	Where to Find Braille Settings	Recommended Braille Table for Code
NVDA	NVDA Menu ▯ Preferences ▯ Settings ▯ Braille	Unified English Braille Grade 1
JAWS	Insert+F3 ▯ Options ▯ Basics ▯ Braille (or Settings Center)	English Braille Grade 1 (uncontracted)
Windows Narrator	Settings ▯ Accessibility ▯ Narrator ▯ Braille	English Grade 1 (uncontracted)
Dolphin SuperNova	CapsLock+SpaceBar ▯ Braille ▯ Display and Translation	Grade 1 (uncontracted)

This page gives a practical overview. Always consult your braille display vendor's documentation and your screen reader's braille guide for device-specific setup steps and advanced configuration options.

Editor Selection and Accessibility Setup Guide

Table of Contents

Editor Comparison

Overview Table

Feature	Notepad	Notepad++	Visual Studio Code
Cost	Free (built-in)	Free	Free
Learning Curve	Minimal	Low	Moderate
Screen Reader Support	Good (basic)	Good (syntax features)	Excellent (built-in accessibility)
Extension/Plugin System	None	Limited	Extensive
Keyboard Navigation	Good	Good	Excellent
Customization	None	Moderate	Very high
Performance	Excellent	Very good	Good
Syntax Highlighting	None	Yes (OpenSCAD available)	Yes (OpenSCAD available)

Feature	Notepad	Notepad++	Visual Studio Code
Terminal Integration	None	None	Built-in
Real-time Feedback	None	None	Yes (with extensions)
Hot Key Customization	Limited	Good	Excellent
File Size Handling	Good	Good	Excellent
Built-in Debugging	None	None	Limited

Detailed Comparison

Notepad Advantages:

- Minimal interface with no distractions-excellent for absolute beginners
- Very predictable behavior for screen reader users
- Extremely fast file operations
- No configuration required; works immediately
- Pure text editing with no formatting surprises

Disadvantages:

- No syntax highlighting (OpenSCAD code appears as plain text)
- No keyboard shortcuts for common editing tasks
- Limited undo/redo capabilities compared to modern editors
- No integrated terminal (requires separate command prompt window)
- No way to run 3dm commands directly

Best For: Users who prefer absolute simplicity and want minimal cognitive load during learning phase

Screen Reader Experience: Windows Narrator reads all content clearly; JAWS and NVDA work well with standard keyboard navigation

Notepad++ Advantages:

- Lightweight and fast
- Good syntax highlighting for OpenSCAD code
- Customizable interface with configurable keyboard shortcuts
- Tab management for multiple files
- Better organization than Notepad for managing projects
- Good screen reader support for basic operations

Disadvantages:

- No built-in terminal (requires external command prompt)

- Plugin system is limited compared to VSCode
- Screen reader experience with syntax highlighting features can be inconsistent
- Less extensive keyboard customization than VSCode
- No integrated development environment features

Best For: Users who want a lightweight editor with syntax highlighting but prefer not to use a full IDE

Screen Reader Experience: JAWS and NVDA handle navigation well; Windows Narrator works but may struggle with complex UI elements

Visual Studio Code Advantages:

- Extensive built-in accessibility features (accessibility inspector, keyboard navigation shortcuts)
- Excellent OpenSCAD extension available (scad-preview)
- Integrated terminal allows running 3dm commands without context switching
- Powerful keyboard shortcut customization
- Rich extension ecosystem for workflow enhancement
- Remote development capabilities
- Native support for multiple projects and workspaces
- Excellent search and find/replace functionality

Disadvantages:

- Steeper learning curve than Notepad or Notepad++
- Requires initial configuration for accessibility
- More resource-intensive than lighter editors
- Built-in terminal can be distracting for some users (alternative: Alt-Tab to standalone terminal)

Best For: Users building comprehensive accessibility workflow and wanting integrated development environment

Screen Reader Experience: Excellent; built specifically with accessibility in mind. NVDA, JAWS, Windows Narrator, and Dolphin all receive high-quality support.

Editor Setup for 3dMake

Setting Default Editor in 3dMake

The 3dMake tool uses your system default text editor. However it can be changed with an edit to the global configuration file by typing this into any terminal.

```
3dm edit-global-config
```

You get a file like this in your default text edit program, typically notepad on Windows.

```
view = "3sil"
model_name = "main"
auto_Start_prints = true
printer_profile = "bambu_labs_X1_carbon"
```

Add one of the following lines to the end of the file:

`editor = "code"` for VSCode `editor = "notepad"` for notepad (already the default) `editor = '''C:\Program Files (x86)\Notepad++\notepad++.exe'''` for Notepad++ (the triple apostrophes mean you do not have to escape the spaces)

Notepad Setup

Configuration Steps:

1. Open 3dMake-generated .scad files directly:
   - Navigate to your project folder in File Explorer
   - Right-click the `.scad` file -> "Open with" -> "Notepad"
   - File opens immediately for editing
2. Edit and Save:
   - Make changes to your code
   - Press `Ctrl+S` to save
   - File updates automatically if 3dMake renders in background
3. Run 3dMake Commands:
   - Save your edits (Ctrl+S)
   - Alt-Tab to Command Prompt or PowerShell window
   - Navigate to project directory: `cd C:\path\to\project`
   - Run 3dMake command: `3dMake render project.scad`

Keyboard Shortcuts:

- `Ctrl+H`: Find and Replace
- `Ctrl+G`: Go to Line (newer versions)
- `Ctrl+Z`: Undo
- `Ctrl+Y`: Redo

Notepad++ Setup

Installation and Configuration:

1. Download from: https://notepad-plus-plus.org/downloads/
2. Choose: Standard Installer (for automatic system integration)

3. Configure OpenSCAD Language Support:

   - Open Notepad++
   - Language -> User Defined Language -> Import… (if OpenSCAD UDL available)
   - Or manually set syntax highlighting:
     - Language -> OpenSCAD (if available in language menu)
     - Otherwise Language -> C++ (provides similar highlighting)

4. Customize for Accessibility:

   - Settings -> Preferences -> General
   - Check: "Minimize to system tray" (optional)
   - Settings -> Preferences -> MISC.
   - Ensure Word Wrap is set to preference
   - Settings -> Preferences -> Backup
   - Enable regular backups of your work

5. Set as Default Editor (see registry method above)

Recommended Keyboard Shortcuts (User-Defined):

Create custom shortcuts by:

- Settings -> Shortcut Mapper
- Add shortcuts for frequently used actions:
  - Save and Switch to Terminal: Alt+T
  - Copy File Path: Ctrl+Shift+C

Tab Management:

- Open multiple files: Each opens in a separate tab
- Switch between tabs: Ctrl+Tab (next) / Ctrl+Shift+Tab (previous)
- Close tab: Ctrl+W

Running 3dMake Commands:

- Save file with Ctrl+S
- Alt-Tab to standalone terminal or command prompt
- Run: `3dMake render filename.scad`

Visual Studio Code Setup

Installation:

1. Download from: https://code.visualstudio.com/ or type `winget search VSCode` in a terminal and then `winget install` whichever option you prefer
2. Install: Run installer and follow prompts
3. Launch: Open VSCode

Enable OpenSCAD Support:

1. Open Extensions (Ctrl+Shift+X)
2. Search: ”OpenSCAD”
3. Install: ”scad-preview” by Antyos
4. Install: ”OpenSCAD Syntax Highlighter” (optional, for better syntax highlighting)

Initial Accessibility Configuration:

1. Open Settings: Ctrl+,
2. Search: ”accessibility”
3. Enable Key Settings:
   - Accessible View: Toggle ON
   - Screen Reader: Select your screen reader (NVDA, JAWS, Narrator, Dolphin)
   - Keyboard Navigation: Ensure enabled
   - Bracket Pair Guides: Can help with code structure understanding
4. Configure Editor Font:
   - Search: ”editor.fontSize”
   - Set to comfortable size (recommend 14-16 for better readability)
   - Search: ”editor.fontFamily”
   - Select monospace font (e.g., ”Consolas” or ”Courier New”)

Set as Default Editor (see registry method above)

VSCode Terminal Options:

Option 1: Using Built-in Terminal (Less Accessible)

1. View -> Terminal (or Ctrl+⌧)
2. Terminal opens at bottom of VSCode window
3. Run commands: `3dMake render filename.scad`
4. Note: Switching focus between editor and terminal requires Tab navigation, which can be cumbersome for screen reader users

Keyboard Navigation:

- Ctrl+⌧ : Toggle terminal visibility
- Ctrl+Shift+⌧ : Create new terminal
- Alt+^/v : Switch between terminals

Option 2: Alt-Tab to Standalone Terminal (RECOMMENDED for Accessibility)

1. Keep Command Prompt/PowerShell open:
   - Open Command Prompt (Win+R, type ”cmd”, Enter)
   - Position window or minimize to taskbar

    - Navigate to project directory: `cd C:\path\to\project`

2. From VSCode, switch terminals:

    - Alt+Tab to Command Prompt
    - Run command: `3dMake render filename.scad`
    - Alt+Tab back to VSCode
    - Continue editing

Why This Is More Accessible:

- Screen reader focus switches clearly between two applications
- Terminal output is read without VSCode context interference
- Cleaner context switching for command-line workflows
- Easier to diagnose issues when editor and terminal are separate

Keyboard Shortcuts for Common Tasks:

Action	Shortcut
Save	Ctrl+S
Find	Ctrl+F
Find and Replace	Ctrl+H
Go to Line	Ctrl+G
Open File	Ctrl+O
Open Folder	Ctrl+K, Ctrl+O
Open Terminal	Ctrl+□
Alt-Tab to Another Window	Alt+Tab
Command Palette	Ctrl+Shift+P

Project Organization in VSCode:

1. Open Project Folder:

    - File -> Open Folder (Ctrl+K, Ctrl+O)
    - Select your project directory
    - All project files appear in Explorer sidebar

2. File Navigation:

    - Press Ctrl+P for Quick Open
    - Type filename to search and jump to file
    - Press Enter to open

3. Quick Switch Between Files:

    - Ctrl+Tab : Open recent files list
    - Arrow keys to select
    - Enter to open

Screen Reader Indent Announcement Configuration

Proper indent announcement is critical for OpenSCAD development, as indentation indicates code nesting and structure.

NVDA (NonVisual Desktop Access)

Enable Indent Announcement:

1. Open NVDA Menu: Alt+N or right-click NVDA icon
2. Preferences -> Settings (Ctrl+Comma)
3. Document Formatting: Tab to it
4. Check: ”Report indentation”
5. In the ”Indentation reporting” dropdown: Select ”Tones and speech”
6. Tone Description: NVDA will announce indent level as progressively higher tones (or speaking indent amount)
7. Apply: Click OK

Additional Tab Stop Configuration:

1. Preferences -> Settings -> Document Formatting
2. Check: ”Report line indentation”
3. This will announce: ”Indent level 4” or similar as you navigate code

Testing:

- Open a `.scad` file with nested code (e.g., difference() { cube(); sphere(); })
- Press Down Arrow to move line by line
- NVDA announces indentation level on each new indented line

Keyboard Control:

- NVDA+3 on Numpad: Cycles indent/outline level reporting

JAWS (Freedom Scientific)

Enable Indent Announcement:

1. Open JAWS Manager: Press JAWSKey+F2 (or right-click JAWS icon)
2. Utilities -> Settings Manager
3. Search: ”Indent”
4. Look for setting: ”Announce Indentation” or ”Report Indentation”
5. Enable: Set to ”Tones” or ”Tones and Speech”
6. Speech Indent Announcement: Speak indent level
7. Tone Indent Announcement: Pitch increases with indent level

Advanced Configuration (Custom Scripts):

If built-in settings don't work:

1. JAWSKey+F2 -> Utilities -> Settings Manager
2. Search: ”Line Breaks” or ”Formatting”

3. Ensure: ”Report line indentation” is enabled
4. Set tone adjustment: Higher pitch for deeper indents

Testing:

- Open `.scad` file with nested code
- Navigate with arrow keys
- JAWS announces indent changes with tones or speech

Keyboard Shortcuts:

- JAWSKey+Alt+I: Toggle indent reporting
- JAWSKey+Alt+Shift+I: Cycle between indent reporting modes (speech/tones/off)

Windows Narrator

Enable Indent Announcement:

1. Open Settings: Win+I
2. Ease of Access -> Narrator
3. Advanced Options: Scroll down
4. Check: ”Report indentation”
5. Indentation Reporting: Select ”Tones” (less intrusive) or ”Speech” (explicit)
6. Apply settings

Narrator Keyboard Shortcuts:

- Narrator+Page Down: Read from current position to end of window
- Narrator+Alt+Arrow Keys: Navigate text
- Narrator+V, I: Customize indentation reporting (in Narrator settings)

Testing:

- Open `.scad` file
- Use Narrator+Page Down to read through code
- Listen for indent tone changes or announcements

Note: Windows Narrator has fewer customization options than JAWS/NVDA; consider NVDA or JAWS for deeper indent control

Dolphin EasyConverter (Dolphin Screen Reader)

Enable Indent Announcement:

1. Open Dolphin Central: Right-click Dolphin icon or click Dolphin icon in taskbar
2. Utilities -> Settings -> Text Processing
3. Look for: ”Indentation” section
4. Enable: ”Announce indentation”
5. Mode: Select ”Tones”, ”Speech”, or ”Tones and Speech”

6. Tone Pitch: Configure pitch increase for deeper indents
7. Apply

ECO (Ease of Cursor Operation) Customization:

1. Dolphin Central -> Utilities -> ECO Settings
2. Text Options -> Indentation Reporting
3. Set preferred announcement style

Testing:

- Open `.scad` file with indented code
- Navigate with arrows
- Dolphin announces indent changes

Keyboard Shortcuts:

- Ctrl+Dolphin+I: Toggle indent reporting on/off
- Ctrl+Dolphin+Shift+I: Cycle indent reporting mode

Curriculum-Specific Editor Recommendations

For Absolute Beginners (Lesson 1-2)

Recommended: Notepad or Notepad++

Rationale:

- Minimal interface reduces cognitive load
- Focus stays on learning OpenSCAD syntax, not editor features
- Keyboard navigation is straightforward
- Screen reader experience is predictable

Setup:

1. Use Notepad or Notepad++ as default editor
2. Configure screen reader indent announcement
3. Keep separate Command Prompt window open for 3dMake commands
4. Alt-Tab workflow between editor and terminal

Workflow Example:

```
1. Open Command Prompt -> Navigate to project folder
2. Run: 3dMake new myproject
3. Alt+Tab to file explorer, open myproject.scad
4. Notepad++ opens file
5. Edit code
6. Ctrl+S to save
7. Alt+Tab to Command Prompt
8. Run: 3dMake render myproject.scad
9. Check output, return to Notepad++ to refine code
```

For Intermediate Users (Lesson 3-6)

Recommended: Notepad++ or VSCode

Notepad++:

- Adds project organization without overwhelming complexity
- Tab support for managing multiple files
- Syntax highlighting improves code understanding
- Still lightweight and predictable

VSCode:

- Opens doors to more sophisticated workflows
- Extension system enables advanced features (OpenSCAD preview)
- Keyboard customization becomes valuable
- Terminal integration useful but use Alt-Tab method

Setup Decision Tree:

- Choose Notepad++ if: You prefer simplicity and want to focus on code logic
- Choose VSCode if: You're ready to invest time learning editor features for long-term benefit

Workflow Example with VSCode:

```
1. Open VSCode (folder view of project)
2. Press Ctrl+P to open file search
3. Type filename and press Enter to open
4. Edit code with autocomplete
5. Ctrl+H for find/replace across project
6. Ctrl+S to save
7. Alt+Tab to Command Prompt
8. Run: 3dMake render filename.scad
9. Alt+Tab back to VSCode to refine code
```

For Advanced Users (Lesson 7-11)

Recommended: Visual Studio Code

Rationale:

- Powerful search/replace across large projects
- Extension system enables specialized workflows
- Keyboard customization reaches full potential
- Workspace management for complex projects
- Debugging capabilities aid troubleshooting

Advanced Setup:

1. Create custom keyboard shortcuts:
    - Ctrl+Alt+R: Save and render current file
    - Ctrl+Alt+P: Preview (if using scad-preview extension)
2. Install Additional Extensions:
    - ”scad-preview”: Real-time 3D preview

   - ”Better Comments”: Categorize comments with colors/tones
   - ”Bracket Pair Colorizer”: Visual/tonal bracket matching
   - ”GitLens”: Track code changes over time

3. Create Task Runner for Common Commands:

   - Ctrl+Shift+B: Configure build task to run 3dMake
   - Create separate tasks for render, export, etc.

4. Use Workspaces:

   - File -> Save Workspace As...
   - Save project-specific workspace with all settings
   - Reopen same workspace configuration automatically

Advanced Workflow Example:

```
1. Open VSCode with project workspace
2. Ctrl+Shift+P -> Run Task -> "3dMake Render Current"
3. Renders file and shows output
4. Use scad-preview extension for real-time 3D view
5. Edit code with advanced search/replace
6. Ctrl+Alt+R saves and renders automatically
7. Use version control (Git) for tracking changes
```

Quick Reference: Editor Comparison for Curriculum

Foundations (Lessons 1-2)

- Primary: Notepad or Notepad++
- Focus: Learn syntax and basic concepts
- Terminal: Standalone Command Prompt (Alt-Tab)

Core Skills (Lessons 3-6)

- Primary: Notepad++ or VSCode
- New Features: Begin using editor syntax highlighting
- Terminal: Standalone (continue Alt-Tab method)
- Skills: File organization, search/replace basics

Advanced Projects (Lessons 7-11)

- Primary: VSCode (strongly recommended)
- Advanced: Use extensions, real-time preview, complex project management
- Terminal: Choose Alt-Tab or built-in based on preference
- Skills: Workspaces, task automation, version control

Troubleshooting Common Issues

Problem: Screen Reader Not Announcing Indent

Solution:

1. Verify indent announcement is enabled in screen reader settings (see above)
2. Test with existing `.scad` file with clear indentation
3. Try different announcement modes (tones vs. speech)
4. Restart screen reader: Alt+Ctrl+N (NVDA) or app restart (JAWS)

Problem: 3dMake Commands Not Running from VSCode Terminal

Solution:

1. Ensure 3dMake is in your system PATH
2. Use standalone terminal instead (Alt-Tab method) - more reliable
3. In VSCode terminal, manually navigate to correct directory first
4. Verify command syntax: `3dMake render filename.scad`

Problem: File Not Saving in Editor

Solution:

1. Verify you pressed Ctrl+S
2. Check file permissions on project folder
3. Try ”Save As” instead
4. Ensure filename includes `.scad` extension

Problem: Syntax Highlighting Not Working

Solution:

1. Verify file has `.scad` extension
2. In Notepad++: Language menu -> select OpenSCAD or C++
3. In VSCode: Install OpenSCAD extension (search Extensions)
4. Restart editor

Problem: Alt-Tab Not Switching Between Windows

Solution:

1. Ensure Command Prompt is open and minimized (not closed)
2. Press Alt+Tab and hold briefly to see window switcher
3. Use Alt+Tab multiple times if more than 2 windows open
4. Alternatively, click taskbar directly (Alt+Tab usually more accessible)

Next Steps

After completing this setup guide:

1. Choose your editor based on the recommendations for your skill level
2. Configure screen reader indent announcement immediately (critical for code structure understanding)
3. Set editor as default for `.scad` files
4. Test with a simple file: Create a test project and edit it
5. Practice Alt-Tab workflow before moving to Lesson 1
6. Document your setup in a personal note for reference

You are now ready to begin Lesson 1: Environmental Configuration and Developer Workflow

Additional Resources

- NVDA Documentation: https://www.nvaccess.org/documentation/
- JAWS Documentation: https://www.freedomscientific.com/products/software/jaws/
- VSCode Accessibility: https://code.visualstudio.com/docs/editor/accessibility
- Windows Narrator Guide: https://support.microsoft.com/en-us/help/22798/windows-11-narrator-get-started
- Notepad++ Documentation: https://notepad-plus-plus.org/online-help/

Command-Line Fundamentals - Choose Your Path

Welcome! Before diving into 3D design with OpenSCAD, you'll master command-line fundamentals. This page will help you understand what command-line interfaces are and choose the best path for you.

What is a Command-Line Interface (CLI)?

A command-line interface is a text-based way to control your computer by typing commands instead of clicking buttons. It's like sending written instructions to your computer.

Why learn it?

- Speed: Text commands are often faster than clicking through menus
- Precision: Exact control over what your computer does
- Accessibility: Perfect for screen readers - text is naturally readable
- Automation: Repeat tasks automatically
- 3D Printing: Essential for batch processing models and integrating tools

Real-world example

Instead of:

1. Opening File Explorer (click)
2. Navigating folders (click, click, click)
3. Right-clicking a file (click)
4. Selecting ”Copy” (click)
5. Navigating to destination (click, click)
6. Right-clicking (click)

7. Selecting ”Paste” (click)

You type: `cp myfile.txt backup/` and press Enter. Done.

Three Command-Line Options on Windows

Windows offers three ways to use the command line. All are accessible with screen readers. Here's how they compare:

Option 1: Windows Command Prompt (CMD)

What it is: The original Windows command-line (1981-present)

Best for: Absolute beginners, maximum simplicity

Pros:

- Simple commands and syntax
- Minimal learning curve
- Easy to understand error messages
- Great for basic file operations
- Perfect entry point to command-line world

Cons:

- Limited advanced features
- Less powerful than alternatives
- No built-in piping (but available)
- Smaller ecosystem

Typical command:

```
copy myfile.txt backup\
```

Option 2: PowerShell

What it is: Microsoft's modern, powerful shell (2006-present)

Best for: Intermediate users, advanced automation

Pros:

- Very powerful for scripting
- Modern syntax and features
- Excellent for 3D printing automation
- Professional workflows
- Large community

Cons:

- Steeper learning curve than CMD
- More complex syntax
- More ”wordy” commands
- Overkill for simple tasks

Typical command:

```
Copy-Item -Path myfile.txt -Destination backup/
```

Option 3: Git Bash

What it is: A Unix/Linux shell on Windows (runs bash inside Git for Windows)

Best for: Programmers, users familiar with Linux, advanced users

Pros:

- Familiar if you know Linux/Unix
- Powerful piping and text processing
- Consistent with other platforms (macOS, Linux)
- Excellent for advanced workflows
- Industry-standard for developers

Cons:

- Requires Git installation
- Steeper learning curve
- Path syntax is different from native Windows
- Less integrated with Windows system tools
- May be ”too much” for beginners

Typical command:

```
cp myfile.txt backup/
```

Command Comparison Table

Here's how common tasks compare across the three options:

Task	Command Prompt	PowerShell	Git Bash
Show current location	`cd`	`pwd`	`pwd`
List files	`dir /B`	`ls -n`	`ls`
Go to folder	`cd Documents`	`cd Documents`	`cd Documents`
Go up one level	`cd ..`	`cd ..`	`cd ..`

Task	Command Prompt	PowerShell	Git Bash
Go home	`cd %USERPROFILE%`	`cd ~`	`cd ~`
Create folder	`mkdir Projects`	`mkdir Projects`	`mkdir Projects`
Create file	`echo text > file.txt`	`echo "text" > file.txt`	`echo "text" > file.txt`
Copy file	`copy old.txt new.txt`	`Copy-Item old.txt new.txt`	`cp old.txt new.txt`
Move file	`move old.txt folder/`	`Move-Item old.txt folder/`	`mv old.txt folder/`
Delete file	`del file.txt`	`Remove-Item file.txt`	`rm file.txt`
List with filter	`dir /B *.txt`	`ls *.txt`	`ls *.txt`
Save output to file	`dir > list.txt`	`ls > list.txt`	`ls > list.txt`
Page through output	`dir | more`	`ls | more`	`ls | less`
Search in files	`findstr "text" file.txt`	`Select-String "text" file.txt`	`grep "text" file.txt`
Show file contents	`type file.txt`	`cat file.txt` or `Get-Content`	`cat file.txt`
Create script	`.bat` files	`.ps1` files	`.sh` files
Run script	`script.bat`	`.\script.ps1`	`./script.sh`

Feature Comparison Table

Feature	CMD	PowerShell	Git Bash
Simplicity	Easiest	Moderate	Hardest
Beginner-Friendly	Best	Good	Challenging
Power/Capability	Basic	Excellent	Excellent
Screen Reader Compatible	Perfect	Perfect	Perfect
Linux/macOS Skills	Windows-only	Some overlap	Full overlap
3D Printing Automation	Adequate	Excellent	Adequate
Learning Curve	Gentle	Moderate	Steep
Community Support	Moderate	Excellent	Excellent
Windows Integration	Perfect	Perfect	Good
Installation Difficulty	Built-in	Built-in	Requires Git

Quick Learner Profile Test

Answer these questions to find your best match:

Question 1: Experience Level

- A: I've never used a command line Easier paths better (CMD or PowerShell)
- B: I've used terminals before Any path works
- C: I use macOS or Linux Git Bash most natural

Question 2: What matters most?

- A: Simplicity and quick learning Choose CMD
- B: Power and advanced features Choose PowerShell
- C: Consistency across Windows/Mac/Linux Choose Git Bash

Question 3: Future goals

- A: Just need to manage files for 3D printing CMD is fine
- B: Advanced automation and scripting PowerShell recommended
- C: Professional development workflows Git Bash best

Question 4: Your main concern

- A: Don't want steep learning curve CMD
- B: Want industry-standard skills Git Bash
- C: Want Microsoft's modern tool PowerShell

Recommendation by Goal

Goal: "I want to learn the basics and get to 3D printing quickly"

Start with CMD (Command Prompt)

- Simplest syntax
- Fastest to get productive
- All core concepts transfer to others
- Can switch later if needed

Start CMD Foundation

Goal: "I want power and professional automation"

Start with PowerShell

- Microsoft's modern, recommended tool
- Professional-grade capabilities
- Better for complex 3D printing workflows
- Skills are in-demand

Start PowerShell Foundation

Goal: ”I want skills that work on Windows, Mac, and Linux”

Start with Git Bash

- Unix/bash skills transfer everywhere
- Great preparation for professional development
- Consistent across all platforms
- Growing standard in 3D printing tools

Start Git Bash Foundation

Can I Switch Paths Later?

Yes, absolutely! All three teach the same fundamental concepts:

- File navigation and organization
- Creating and managing files/folders
- Combining commands for powerful workflows
- Scripting and automation basics

Once you learn one, switching to another is quick. The concepts are identical; only the syntax changes.

Example: If you learn CMD first, then later want PowerShell's power, you'll find it easy. The command `cd Documents` works the same way in all three.

Important: All Are Equally Accessible

Screen readers work perfectly with all three

- Text-based by nature (perfect for NVDA, JAWS)
- No mouse required
- Output is naturally readable
- Keyboard-only workflows

Don't let accessibility concerns influence your choice. All are fully accessible.

Getting Started: Your Decision

Take a moment and choose:

1. I want the simplest path

Command Prompt Foundation

- Time to first success: ~30 minutes
- Learning curve: Gentlest
- When to upgrade: Once you're comfortable and want power

2. I want modern, powerful Windows tools

PowerShell Foundation

- Time to first success: ~45 minutes
- Learning curve: Moderate
- Best for: Professional automation, 3D printing workflows

3. I want Unix/Linux skills that work everywhere

Git Bash Foundation

- Time to first success: ~1 hour
- Learning curve: Steeper but rewarding
- Best for: Professional development, cross-platform work

Not Sure? Here's What Most People Do

If you're reading this and unsure:

> Start with Command Prompt (CMD). It's the gentlest introduction. You'll be productive quickly and can always switch to PowerShell or Git Bash later. The skills transfer completely.

After completing CMD:

- Want more power? PowerShell is next
- Want Linux skills? Git Bash is next
- Want to stick with CMD? You have all the skills you need

FAQ

Q: Do I need to pick now and stick with it forever? A: No. Start with one, try another, switch between them. They're tools. Use what works.

Q: Will my 3D printing skills work in all three? A: Yes. Once you understand the concepts (file organization, automation, piping), they apply everywhere.

Q: If I pick CMD, can I learn PowerShell later? A: Absolutely. Many learners do exactly this. CMD gets you productive; PowerShell adds power.

Q: Is Git Bash harder? A: Slightly, due to path syntax and Unix conventions. But not dramatically. If you take time with it, you'll learn it.

Q: Which do professional 3D printing developers use? A: Mix of all three, but Git Bash/Linux is most common in cross-platform teams.

Ready to Begin?

Choose your path above and click to start. Remember:

- Each lesson includes practice exercises
- You can't break anything
- Mistakes are learning opportunities
- Ask for help if stuck

Let's get you comfortable with the command line!

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

PowerShell

This section covers terminal fundamentals, screen reader accessibility, and command-line basics needed before diving into 3D design with OpenSCAD.

Time commitment: ~10 hours

Skills gained: Terminal navigation, file operations, basic scripting, keyboard-only workflow mastery

PowerShell for Screen Reader Users - Complete Curriculum Overview

Welcome! This curriculum teaches you how to use PowerShell (Windows Terminal) as a screen reader user, starting from zero experience and building to professional-level skills.

Last Updated: February 2026
Total Duration: 30-45 hours of instruction + practice (for screen reader users)
Target Users: Anyone with a screen reader (NVDA, JAWS, or other)

Note: Time estimates reflect the additional time needed for screen reader navigation, text-to-speech processing, and careful keyboard-based workflows.

Why Learn PowerShell?

For Everyone

- Speed: Text commands are often faster than clicking through menus
- Automation: Repeat tasks automatically instead of doing them manually
- Precision: Exact control over what your computer does
- Scripting: Create programs that solve real problems

For 3D Printing (Our Focus)

- Batch Operations: Process 100s of 3D models at once
- Accessibility: Many 3D design tools are scriptable
- Reproducibility: Same settings, every time
- Integration: Connect OpenSCAD, slicers, and tools together

For Screen Reader Users Specifically

- Great Accessibility: PowerShell works perfectly with NVDA, JAWS, and others
- No Mouse Needed: Everything is keyboard-based
- Text-Based: Output is naturally readable by screen readers
- Stability: Unlike GUIs, terminal interactions are consistent

Curriculum Structure

Phase 1: Absolute Beginner -> Comfortable User

Lesson	Duration	What You'll Learn
Screen Reader Accessibility Guide	1.5 hours	Screen reader tips specific to PowerShell (READ FIRST)
PS-Pre: Your First Terminal	2-2.5 hours	Opening PowerShell, first commands, basic navigation
PS-0: Getting Started	1.5 hours	Paths, shortcuts, tab completion
PS-1: Navigation	2-2.5 hours	Moving around the file system confidently

Goal: You can navigate to any folder and see what's in it with your screen reader.

Phase 2: Intermediate User -> Power User

Lesson	Duration	What You'll Learn
PS-2: File & Folder Manipulation	2.5-3 hours	Create, copy, move, delete files/folders
PS-3: Input, Output & Piping	2.5-3 hours	Chain commands together, redirect output
PS-4: Environment Variables & Aliases	2-2.5 hours	Automate settings, create shortcuts

Lesson	Duration	What You'll Learn
PS-5: Filling in the Gaps	2-2.5 hours	Profiles, history, debugging

Goal: You can create folders, manage files, and combine commands to accomplish complex tasks.

Phase 3: Professional Skills (Beyond Curriculum) These topics extend beyond this curriculum but are natural next steps:

Topic	When to Learn
Scripting (.ps1 files)	After PS-5
Functions & Loops	After PS-5
Error Handling	After PS-5
Remote Administration	Advanced
3D Printing Integration	After all above

How to Use This Curriculum

If You've Never Used a Terminal Before Start here and go in order:

1. Read Screen Reader Accessibility Guide completely
2. Do PS-Pre: Your First Terminal exercises
3. Continue with PS-0, PS-1, etc.

Don't skip steps - each builds on the previous one.

If You've Used a Terminal Before (But Not with a Screen Reader) Start here:

1. Skim Screen Reader Accessibility Guide (you'll recognize most tips)
2. Quickly review PS-Pre (basics with screen reader focus)
3. Move to PS-0 for deeper learning

If You're Experienced with Terminal + Screen Reader You can:

1. Jump to specific lessons you need (PS-2, PS-3, etc.)
2. Use the Quick Reference sections
3. Skip the practice exercises, do the quizzes to verify knowledge

How Each Lesson is Structured

Every Lesson Contains:

1. Learning Objectives - What you'll be able to do

2. Key Commands - The important ones to memorize
3. Step-by-Step Examples - How to actually do it
4. Practice Exercises - Hands-on work
5. Quiz Questions - Check your understanding
6. Extension Problems - Go deeper if interested

How to Get Through Each Lesson:

1. Read the learning objectives
2. Do the step-by-step examples alongside
3. Complete the practice exercises (critical!)
4. Take the quiz (don't cheat)
5. Try extension problems if you have time
6. Move to next lesson when quiz is solid

Estimated time: 2-3 hours per lesson for screen reader users (depends on practice time)

Screen Reader Tips Throughout the Curriculum

Every Lesson Includes:

- [SR] symbols marking screen reader-specific sections
- Tips for NVDA and JAWS users separately
- Solutions for common accessibility issues
- Workarounds for long outputs

Screen Reader Accessibility Guide This is your companion resource used throughout:

- Detailed NVDA keyboard shortcuts
- Detailed JAWS keyboard shortcuts
- Solutions to common problems
- Pro tips for efficiency

Keep it open or printed as you work through lessons.

Quick Start Guide (First 45-60 Minutes)

If You Have 45-60 Minutes Right Now:

1. Open PowerShell (Windows key -> type PowerShell -> Enter)
2. Run these commands:

```
pwd
ls -n
cd Documents
pwd
```

3. See how your screen reader reads each output

4. Try Tab completion:
    - Type `cd D` and press Tab
    - Hear PowerShell auto-complete to Documents (or similar)
5. Create a file:

```
echo "I am learning PowerShell" > learning.txt
cat learning.txt
```

That's it! You've done the key concepts. Now read PS-Pre for the details.

Common Questions Before Starting

Q: Do I have to use PowerShell? What about Command Prompt (cmd.exe)?
A: Command Prompt works, but PowerShell is better. PowerShell is:

- More powerful
- Better for modern tools
- Screen-reader-friendly
- The future of Windows automation

Use PowerShell for this curriculum.

Q: What if I use a different screen reader (not NVDA or JAWS)? A: The fundamentals work the same. Check your screen reader's documentation for the equivalent of these commands:

- Read current line
- Read to end of screen
- Read next/previous page

Most screen readers have these features.

Q: I'm intimidated. Is this really for me? A: YES. This curriculum is specifically designed for people with no terminal experience AND with screen readers. You'll start with absolute basics. There's nothing to be afraid of - we've written this specifically to make it accessible.

Q: How long will this take? A: Realistically:

- Minimum (just lessons, no exercises): 15-18 hours
- Normal (lessons + exercises): 30-45 hours
- With extension problems: 45-60+ hours

Spread it over weeks or months. Go at your pace.

Q: What if I forget things? A: That's normal and expected. Solutions:

1. Come back to this page for the overview
2. Jump back to that lesson for a quick review

3. Use the quiz questions to self-test
4. Check the Screen Reader Accessibility Guide for troubleshooting

Q: Will this help me with 3D printing? A: Absolutely. Near the end of the 3dMake curriculum, you'll use PowerShell to:

- Batch-process 3D models
- Automate slicing tasks
- Run scripts that generate designs
- Integrate tools together

Suggested Study Schedule

Beginner Goal (Weeks 1-2) Week 1:

- Day 1: Read Screen Reader Accessibility Guide
- Day 2: PS-Pre lesson
- Day 3: PS-0 lesson
- Day 4: Practice PS-0 and PS-1 exercises
- Day 5: PS-1 lesson

Week 2:

- Review PS-0 and PS-1 quizzes
- Practice navigation exercises daily
- Do extension problems for PS-0 and PS-1
- Feel confident with file system navigation

Goal: Know how to navigate to any folder, list its contents, and understand paths.

Intermediate Goal (Weeks 3-5) Week 3:

- PS-2 lesson (file manipulation)
- Complete exercises
- Take quiz

Week 4:

- PS-3 lesson (piping and output)
- Complete exercises
- Take quiz

Week 5:

- PS-4 and PS-5 lessons
- Complete all quizzes
- Practice combining commands

Goal: Create, modify, and move files. Combine commands for complex tasks.

Advanced Goal (Weeks 6+)

- Review any lessons you need
- Do all extension problems
- Start learning PowerShell scripting
- Begin 3D printing integration

Success Criteria

By the End of PS-1, You Should:

- ☐ Know where you are at all times (`pwd`)
- ☐ See what's around you (`ls -n`)
- ☐ Navigate confidently with your screen reader
- ☐ Use Tab completion comfortably
- ☐ Understand absolute and relative paths
- ☐ Pass the PS-0 and PS-1 quizzes

By the End of PS-3, You Should:

- ☐ Create and delete files and folders
- ☐ Copy and move files
- ☐ Redirect output to files
- ☐ Pipe commands together
- ☐ Save long outputs to readable files
- ☐ Pass all quizzes PS-0 through PS-3

By the End of PS-5, You Should:

- ☐ Use command history effectively
- ☐ Create aliases and functions
- ☐ Understand your PowerShell profile
- ☐ Handle screen reader edge cases
- ☐ Feel comfortable experimenting
- ☐ Pass all quizzes

Important Rules

Rule 1: Always Know Where You Are Every session, first thing:

```
pwd
```

If you don't know your path, you'll get lost. Don't move until you know where you are.

Rule 2: Check Before You Delete Before deleting anything:

```
ls -n
```

Make sure you're deleting the right thing. Once it's gone, it's gone.

Rule 3: Use `-n` with ls Always:

```
ls -n
```

Never:

```
ls
```

The `-n` (names only) is screen reader friendly. The default view is hard to read.

Rule 4: When Lost, Redirect to a File If output is confusing:

```
command-name > output.txt
notepad.exe output.txt
```

This is always clearer for screen readers than terminal output.

Rule 5: Save Everything You Create Every exercise, save your work:

```
mkdir my-practice-folder
cd my-practice-folder
```

Create a ”learning” folder and keep everything there.

Troubleshooting: ”Nothing Works!”

If you're stuck:

1. Can't hear PowerShell at all?
   - Make sure screen reader is running BEFORE PowerShell
   - Try Alt+Tab to cycle to PowerShell window
   - Restart both screen reader and PowerShell
2. Commands not working?
   - Check spelling carefully
   - Make sure you pressed Enter
   - Try `Get-Help command-name`
3. Can't read the output?
   - Redirect to file: `command > output.txt`
   - Open in Notepad: `notepad.exe output.txt`
   - This always works
4. Something ran forever?
   - Press Ctrl+C to stop it

5. Completely confused?
    - Go back to PS-Pre and start over
    - Work through every single exercise slowly
    - Ask for help (ask an instructor or peer)

Resources

Official Documentation

- PowerShell Docs: https://docs.microsoft.com/powershell/
- Windows Terminal Docs: https://docs.microsoft.com/windows/terminal/

Screen Reader Guides

- NVDA: https://www.nvaccess.org/documentation/
- JAWS: https://www.freedomscientific.com/support/

Learning Resources

- Microsoft Learn: https://learn.microsoft.com/en-us/training/modules/
- PowerShell ISE: Built-in editor (open with `ise`)

3D Printing Integration

- OpenSCAD Scripting: See 3dMake lessons in this curriculum
- Batch Processing: See PS-3 and PS-5 for real examples

Getting Help

If You're Stuck:

1. Read the relevant section of Screen Reader Accessibility Guide
2. Try a different approach from the "Solutions" sections
3. Go back one lesson and strengthen those concepts
4. Ask an instructor or peer with specific questions

Before You Ask for Help, Prepare:

1. What command are you running?
2. What do you expect to happen?
3. What actually happened?
4. What error message did you hear?

Example: "I ran `cd Desktop` but my screen reader said 'Cannot find path'. I expected to go to the Desktop folder."

This helps others help you quickly.

Next Steps

1. Right now: Read the Screen Reader Accessibility Guide completely
2. Next session: Start PS-Pre and do all exercises
3. Keep going: One lesson per day/week at your pace
4. Practice: Do exercises, not just read
5. Check yourself: Take the quizzes honestly
6. Celebrate: Each lesson completed is a real skill gained

Final Thoughts

Learning to use PowerShell with a screen reader is absolutely achievable. Many people do it successfully. This curriculum was designed based on real experiences of screen reader users.

You've got this. Start with PS-Pre, take it slow, do the exercises, and ask questions when stuck.

Welcome to the PowerShell community!

Curriculum Map (For Reference)

```
START HERE v
+---- Screen Reader Accessibility Guide (reference throughout)
+---- PS-Pre: Your First Terminal (absolute beginner entry point)
+---- PS-0: Getting Started (paths & navigation foundations)
+---- PS-1: Navigation (comfortable moving around)
+---- PS-2: File & Folder Manipulation (create/move/delete)
+---- PS-3: Input, Output & Piping (chain commands)
+---- PS-4: Environment Variables & Aliases (automation)
+---- PS-5: Filling in the Gaps (profiles & history)
+---- PS_Unit_Test (comprehensive practice & assessment)
       v
   NEXT: 3D Printing Integration Lessons
       v
   ADVANCED: PowerShell Scripting
```

Questions? Feedback? Stuck? Refer back to this page and the Screen Reader Accessibility Guide. You've got everything you need.

Now open PS-Pre and let's get started!

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

PS-Pre: Your First Terminal - Screen Reader Navigation Fundamentals

Duration: 1.5-2 hours (for screen reader users)
Prerequisites: None - this is the starting point

Accessibility Note: This lesson is designed specifically for screen reader users (NVDA, JAWS)

What is a Terminal?

A terminal (also called a command line or shell) is a text-based interface where you type commands instead of clicking buttons. Think of it like sending written instructions to your computer instead of pointing and clicking.

Why learn this?

- Faster and more precise work (especially for 3D printing scripts and automation)
- Essential for programming and using tools like OpenSCAD
- Accessibility: Many command line tools work perfectly with screen readers
- Scripting: Automate repetitive tasks

Opening PowerShell for the First Time

On Windows Method 1: Search (Easiest)

1. Press the Windows key alone
2. You should hear "Search"
3. Type: `PowerShell`
4. You'll hear search results appear
5. Press Enter to open the first result (Windows PowerShell)
6. PowerShell will open in a new window

Method 2: Using the Start Menu

1. Press Windows key + X (opens the Quick Link menu)
2. Look for "Windows PowerShell" or "Terminal"
3. Press Enter

Method 3: From File Explorer

1. Open File Explorer
2. Navigate to any folder
3. In the menu bar, select "File" -> "Open Windows PowerShell here"
4. PowerShell opens in that folder location

First Connection: Understanding the Prompt When PowerShell opens, your screen reader will announce the window title and then the prompt. The prompt is where you type commands.

What you'll hear:

```
PS C:\Users\YourName>
```

What this means:

- `PS` = ”PowerShell” indicator
- `C:\Users\YourName` = Your current location (the ”path”)
- > = The prompt is ready for your input

Important

Your cursor is blinking right after the >. This is where you type.

Your First Commands (Screen Reader Edition)

Command 1: ”Where Am I?” - `pwd` What it does: Tells you your current location

Type this:

```
pwd
```

Press Enter

What you'll hear: Your screen reader will announce the current path, something like:

```
C:\Users\YourName
```

Understanding paths:

- Paths show your location in the file system (like a mailing address)
- Windows paths use backslashes: `C:\Users\YourName\Documents`
- Think of it like folders inside folders: `C:\` (main drive) -> `Users` -> `YourName` -> `Documents`

Command 2: ”What's Here?” - `ls -n` What it does: Lists all files and folders in your current location. The `-n` flag makes it screen-reader friendly (names only, one per line)

Type this:

```
ls -n
```

Press Enter

What you'll hear: Your screen reader will announce each file and folder name, one per line:

```
Desktop
Documents
Downloads
Music
Pictures
...
```

Why `-n`?

- Without `-n`, PowerShell shows files in columns (hard to read with a screen reader)
- With `-n`, each file/folder is on its own line (perfect for screen readers)

Command 3: ”Go There” - `cd Documents` What it does: Changes your location (navigates to a folder)

Type this:

```
cd Documents
```

Press Enter

What you'll hear: The prompt changes to show your new location. You might hear something like:

```
PS C:\Users\YourName\Documents>
```

Practice navigation:

1. Run `pwd` to confirm you're in Documents
2. Run `ls -n` to see what files are in Documents
3. Try going back: `cd ..` (the `..` means ”go up one level”)
4. Run `pwd` again to confirm
5. Go back to Documents: `cd Documents`

Reading Screen Reader Output (Critical Skills)

Dealing with Long Lists When you run `ls -n` in a folder with many files, the list might be very long. Your screen reader might announce 50+ items rapidly.

Solution 1: Save to a File

```
ls -n > list.txt
notepad.exe list.txt
```

This saves the list to a file and opens it in Notepad where you can read it more slowly.

Solution 2: Search Within the Output

```
ls -n | findstr "search-term"
```

Example: If you're looking for files containing ”scad”, type:

```
ls -n | findstr "scad"
```

Navigating Tab Completion One of the most powerful screen reader tricks is Tab completion:

How it works:

1. Type the first few letters of a folder or file name
2. Press Tab
3. PowerShell automatically completes the rest

Example:

1. You're in `C:\Users\YourName>`
2. Type: `cd Doc`
3. Press Tab
4. PowerShell auto-completes it to: `cd Documents`
5. Press Enter to go there

With a screen reader:

1. As you type `Doc`, your screen reader announces each letter
2. When you press Tab, PowerShell types the rest and your screen reader announces the full command
3. This is much faster than typing the whole thing

Creating and Editing Files

Create a Simple File

Type this:

```
echo "Hello, PowerShell!" > hello.txt
```

What this does:

- `echo` sends text to the screen (or file)
- `"Hello, PowerShell!"` is the text
- > redirects it to a file called `hello.txt`

Read the File Back

Type this:

```
cat hello.txt
```

What you'll hear: Your screen reader announces:

```
Hello, PowerShell!
```

Open and Edit the File

Type this:

```
notepad.exe hello.txt
```

This opens the file in Notepad where you can edit it with your screen reader.

Essential Keyboard Shortcuts

These work in PowerShell and are crucial for screen reader users:

Key Combination	What It Does
Up Arrow	Shows your previous command (press again to go further back)
Down Arrow	Shows your next command (if you went back)
Tab	Auto-completes folder/file names
Ctrl+C	Stops a running command
Ctrl+L	Clears the screen
Enter	Runs the command

Screen reader tip: These all work perfectly with your screen reader. Try them!

Screen Reader-Specific Tips

NVDA Users

1. Reading Command Output:
    - Use NVDA+Home to read the current line
    - Use NVDA+Down Arrow to read to the end of the screen
    - Use NVDA+Page Down to read the next page
2. Reviewing Text:
    - Use NVDA+Shift+Page Up to review text above

JAWS Users

1. Reading Output:
    - Use Insert+Down Arrow to read line-by-line
    - Use Insert+Page Down to read by page
    - Use Insert+End to jump to the end of text
2. Reading All Text:
    - Use Insert+Down Arrow repeatedly
    - Or use Insert+Ctrl+Down to read to the end

Common Issue: "I Can't Hear the Output" Problem: You run a command but don't hear the output

Solutions:

1. Make sure your cursor is at the prompt (try pressing End or Ctrl+End)
2. Use Up Arrow to go back to your previous command and review it
3. Try redirecting to a file: `command > output.txt` then open the file
4. In NVDA: Try pressing NVDA+F7 to open the Review Mode viewer

Practice Exercises

Complete these in order. Take your time with each one:

Exercise 1: Basic Navigation

1. Open PowerShell
2. Run `pwd` and note your location
3. Run `ls -n` and listen to what's there
4. Try `cd Documents` or another folder
5. Run `pwd` to confirm your new location
6. Run `ls -n` in this new location

Goal: You should be comfortable knowing where you are and what's around you

Exercise 2: Using Tab Completion

1. In your home directory, type `cd D` (just the letter D)
2. Press Tab
3. PowerShell should auto-complete to a folder starting with D
4. Repeat with other folder names
5. Try typing a longer name: `cd Down` and Tab to `Downloads`

Goal: Tab completion should feel natural

Exercise 3: Creating and Viewing Files

1. Create a file: `echo "Test content" > test.txt`
2. View it: `cat test.txt`
3. Create another: `echo "Line 2" > another.txt`
4. List both: `ls -n`

Goal: You understand create, view, and list operations

Exercise 4: Going Up Levels

1. Navigate into several folders: `cd Documents`, then `cd folder1`, etc.
2. From deep inside, use `cd ..` multiple times to go back up
3. After each `cd ..`, run `pwd` to confirm your location

Goal: You understand relative navigation with `..`

Exercise 5: Redirecting Output

1. Create a list: `ls -n > directory_list.txt`
2. Open it: `notepad.exe directory_list.txt`
3. Read it with your screen reader
4. Close Notepad
5. Verify the file exists: `ls -n | findstr "directory"`

Goal: You can save long outputs to files for easier reading

Checkpoint Questions

After completing this lesson, you should be able to answer:

1. What does `pwd` do?
2. What does `ls -n` do?
3. Why do we use `-n` with `ls`?
4. What path are you in right now?
5. How do you navigate to a new folder?
6. How do you go up one level?
7. What's the Tab key for?
8. What does `echo "text" > file.txt` do?
9. How do you read a file back?
10. How do you stop a command that's running?

You should be able to answer all 10 with confidence before moving to PS-0.

Common Questions

Q: Do I need to use PowerShell? Can I use Command Prompt (cmd.exe)? A: PowerShell is more powerful and works better with modern tools. We recommend PowerShell, but Command Prompt basics are similar.

Q: Why is my screen reader not reading the output? A: This is common. Use `command > file.txt` to save output to a file, then open it with Notepad for reliable reading.

Q: What if I type something wrong? A: Just press Enter and you'll see an error message. Type the correct command on the next line. No harm done!

Q: How do I get help with a command? A: Type `Get-Help command-name` (we'll cover this in PS-0)

Q: Can I make PowerShell more accessible? A: Yes! We'll cover customization in PS-5.

Next Steps

Once you're comfortable with these basics:

- Move to PS-0: Getting Started for deeper path understanding
- Then continue through PS-1 through PS-5 for full terminal mastery

Resources

- Microsoft PowerShell Docs: https://docs.microsoft.com/powershell/
- NVDA Screen Reader: https://www.nvaccess.org/

- JAWS Screen Reader: https://www.freedomscientific.com/products/software/jaws/
- Windows Terminal Accessibility: https://docs.microsoft.com/windows/terminal/

Troubleshooting

Issue	Solution
PowerShell won't open	Try searching Windows, or right-click a folder and select "Open PowerShell here"
Can't hear the output	Try redirecting to a file: `command > output.txt`
Tab completion not working	Make sure you typed at least one character before pressing Tab
Command not found	Make sure you spelled it correctly; try `Get-Command` to see available commands
Stuck in a command	Press Ctrl+C to stop it

Still stuck? The checkpoint questions and exercises are your best teacher. Work through them multiple times until comfortable.

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

PS-0: Getting Started - Layout, Paths, and the Shell

Estimated time: 20-30 minutes

Learning Objectives

- Launch PowerShell and locate the prompt
- Understand path notation and shortcuts (`~`, `./`, `../`)
- Use tab completion to navigate quickly

Materials

- Computer with PowerShell
- Editor (Notepad/VS Code)

Step-by-step Tasks

1. Open PowerShell and note the prompt (it includes the current path).
2. Run `pwd` and say or note the printed path.
3. Use `ls -n` to list names in your home directory.
4. Practice `cd Documents`, `cd ../` and `cd ~` until comfortable.
5. Try tab-completion: type `cd ~/D` and press Tab.

Checkpoints

- Confirm you can state your current path and move to `Documents`.

Quiz - Lesson PS.0

1. What is a path?
2. What does ~ mean?
3. How do you autocomplete a path?
4. How do you go up one directory?
5. What command lists only names (`ls` flag)?
6. True or False: On Windows, PowerShell uses backslashes (\) in paths, but forward slashes (/) are also accepted.
7. Explain the difference between an absolute path and a relative path.
8. If you are in `C:\Users\YourName\Documents` and you type `cd ../`, where do you end up?
9. What happens when you press Tab while typing a folder name in PowerShell?
10. Describe a practical reason why understanding paths is important for a 3D printing workflow.
11. What does `./` mean in a path, and when would you use it?
12. If a folder path contains spaces (e.g., `Program Files`), how do you navigate to it with `cd`?
13. Explain what the prompt `PS C:\Users\YourName>` tells you about your current state.
14. How would you navigate to your home directory from any location using a single command?
15. What is the advantage of using relative paths (like `../`) versus absolute paths in automation scripts?

Extension Problems

1. Create a nested folder and practice `cd` into it by typing partial names and using Tab.
2. Use `ls -n -af` to list only files in a folder.
3. Save `pwd` output to a file and open it in Notepad.
4. Try `cd` into a folder whose name contains spaces; observe how quotes are handled.

5. Create a short note file and open it from PowerShell.
6. Build a folder structure that mirrors your project organization; navigate to each level and document the path.
7. Create a script that prints your current path and the total number of files in it; run it from different locations.
8. Investigate the special paths (e.g., `$HOME`, `$PSScriptRoot`); write down what each contains and when you'd use them.
9. Compare absolute vs. relative paths by navigating to the same folder using each method; explain which is easier for automation.
10. Create a PowerShell function that changes to a frequently-used folder and lists its contents in one command; test it from different starting locations.
11. Navigate to three different locations and at each one note the prompt, the path from `pwd`, and verify you understand what each shows.
12. Create a complex folder tree (at least 5 levels deep) and navigate it using only relative paths; verify your location at each step.
13. Document all shortcuts you know (`~`, `./`, `../`, `$HOME`) and demonstrate each one works as expected.
14. Write a guide for a peer on how to understand the PowerShell prompt and path notation without using GUI file explorer.
15. Create a troubleshooting flowchart: if someone says "I don't know where I am," what commands do you give them to find out?

References

- Microsoft. (2024). *PowerShell scripting overview and documentation*. https://learn.microsoft.com/powershell/scripting/overview
- Microsoft. (2024). *Filesystem navigation in PowerShell*. https://learn.microsoft.com/powershell/scripting/learn/shell/navigate-the-filesystem
- Microsoft. (2024). *Accessibility features in PowerShell ISE*. https://learn.microsoft.com/powershell/scripting/windows-powershell/ise/accessibility-in-windows-powershell-ise

Helpful Resources

- PowerShell Basics - Microsoft Learn
- Filesystem Navigation Guide
- Understanding Path Notation
- Tab Completion Reference
- Accessibility in PowerShell ISE

PS-1: Navigation - Moving Around Your File System

Duration: 1 class period
Prerequisite: PS-0 (Getting Started)

Learning Objectives By the end of this lesson, you will be able to:

- Use `pwd` to print your current location
- Use `cd` to move between directories
- Use `ls` (and its flags) to list files and folders
- Use wildcards `*` and `?` to filter listings
- Navigate relative vs. absolute paths
- Search for files by name and extension

Materials

- PowerShell
- Text editor (Notepad or VS Code)

Commands Covered in This Lesson

Command	What It Does
`pwd`	Print Working Directory - shows where you are
`cd path`	Change Directory - move to a new location
`ls`	List - shows files and folders in current location
`ls -n`	List names only (screen reader friendly)
`ls -n -af`	List names of files only
`ls -n -ad`	List names of directories only
`ls *.extension`	List files matching a pattern

`pwd` - Where Am I?

Type `pwd` and press `Enter`. PowerShell prints the full path to your current location.

```
pwd
# Output: C:\Users\YourName
```

When to use: Always run this if you're unsure of your current location.

`cd` - Changing Directories

`cd` stands for ”change directory.”

```
# Go to Documents
cd Documents
# Go up one level to parent directory
cd ..
# Go to home directory
cd ~
# Go to a specific path
cd C:\Users\YourName\Documents\3D_Projects
```

`ls` - Listing Files and Folders

Use `ls -n` for screen reader compatibility.

```
# List all files and folders (names only)
ls -n
# List only files (no folders)
ls -n -af
# List only folders (no files)
ls -n -ad
```

Wildcards - Finding Files by Pattern

Wildcards help you find files without typing the full name.

`*` (asterisk) matches any number of characters:

```
# List all .scad files
ls -n *.scad
# List all files starting with "part"
ls -n part*
# List all files ending with "_final"
ls -n *_final*
```

`?` (question mark) matches exactly one character:

```
# Find files like model1.scad, model2.scad (but not model12.scad)
ls -n model?.scad
```

Step-by-step Practice

1. Run `pwd` and confirm your location
2. Move to `Documents`: `cd Documents`
3. Confirm you moved: `pwd`
4. List files and folders: `ls -n`
5. List only files: `ls -n -af`
6. Go back up: `cd ..`
7. Search for files: `ls -n *.txt`

Checkpoints

After this lesson, you should be able to:

- ☐ Navigate to any folder using `cd`
- ☐ Confirm your location with `pwd`
- ☐ List files and folders with `ls -n`
- ☐ Use wildcards to find files by pattern
- ☐ Move between absolute and relative paths confidently

Quiz - Lesson PS.1

1. What does `pwd` show?
2. How do you list directories only with `ls`?
3. What wildcard matches any number of characters?
4. How do you list files with the `.scad` extension?

5. Give an example of an absolute path and a relative path.
6. True or False: The `*` wildcard matches exactly one character.
7. Explain the difference between `ls -n` and `ls -n -ad`.
8. Write a command that would list all `.txt` files in your Documents folder using a wildcard.
9. How would you search for files containing ”part” in their name across multiple files?
10. Describe a practical scenario where using wildcards saves time in a 3D printing workflow.
11. What happens when you use `ls -n part?.scad` versus `ls -n part*.scad`?
12. How would you navigate to a folder whose name contains both spaces and special characters?
13. If you're in `/Documents/Projects/3D` and you want to go to `/Documents/Resources`, what command would you use?
14. Write a command sequence that navigates to the Downloads folder, lists only files, then returns to home.
15. Explain the purpose of using `ls -n -af` specifically in a screen reader context.

Extension Problems

1. Write a one-line script that lists `.scad` files and saves to `scad_list.txt`.
2. Use `ls -n ~/Documents | more` to page through long listings.
3. Combine `ls` with `Select-String` to search for a filename pattern.
4. Create a shortcut alias in the session for a long path and test it.
5. Practice tab-completion in a directory with many similarly named files.
6. Build a PowerShell script that recursively lists all `.scad` and `.stl` files in a directory tree; save the results to a file.
7. Compare the output of `ls`, `Get-ChildItem`, and `gci` to understand PowerShell aliasing; document what each command does.
8. Create a filtering command that displays only files modified in the last 7 days; test it on your documents folder.
9. Write a non-visual guide to PowerShell navigation; include descriptions of common patterns and how to verify directory contents audibly.
10. Develop a navigation workflow for a typical 3D printing project: move between CAD, slicing, and print-log folders efficiently; document the commands.
11. Create a complex wildcard search: find all files in a folder and subfolders that match multiple patterns (e.g., `*_v1.* OR *_final.*`).
12. Build a script that navigates through a folder tree, counts files at each level, and reports the structure.
13. Document the output differences between `ls -n`, `ls -n -af`, `ls -n -ad`, and `Get-ChildItem`; explain when to use each.
14. Create a navigation ”cheat sheet” as a PowerShell script that prints common paths and how to navigate to them.

15. Design a project folder structure on your computer, document each path, then create a script that validates all folders exist.

References

- Microsoft. (2024). *Get-ChildItem cmdlet reference*. https://learn.microsoft.com/powershell/module/microsoft.powershell.management/get-childitem
- Microsoft. (2024). *PowerShell wildcards and filtering*. https://learn.microsoft.com/powershell/scripting/learn/shell/using-wildcards
- Microsoft. (2024). *Navigation best practices in PowerShell*. https://learn.microsoft.com/powershell/scripting/learn/shell/navigate-the-filesystem

Helpful Resources

- Get-ChildItem Cmdlet Reference
- PowerShell Wildcards and Filtering
- Navigation Best Practices
- Relative and Absolute Paths
- Screen Reader Tips for PowerShell

PS-2: File and Folder Manipulation

Estimated time: 30-45 minutes

Learning Objectives

- Create, copy, move, and delete files and folders from PowerShell
- Use `ni`, `mkdir`, `cp`, `mv`, `rm`, and `rmdir` safely
- Understand when operations are permanent and how to confirm results

Materials

- PowerShell
- Small practice folder for exercises

Step-by-step Tasks

1. Create a practice directory: `mkdir ~/Documents/PS_Practice` and `cd` into it.
2. Create two files: `ni file1.txt` and `ni file2.txt`.
3. Copy `file1.txt` to `file1_backup.txt` with `cp` and confirm with `ls -n`.
4. Rename `file2.txt` to `notes.txt` using `mv` and confirm.
5. Delete `file1.txt` with `rm` and verify the backup remains.

Checkpoints

- After step 3 you should see both the original and the backup file.

Quiz - Lesson PS.2

1. How do you create an empty file from PowerShell?
2. What command copies a file?
3. How do you rename a file?
4. What does `rm -r` do?
5. Why is `rm` potentially dangerous?
6. True or False: `cp` requires the `-r` flag to copy both files and folders.
7. Explain the difference between `rm` and `rmdir`.
8. If you delete a file with `rm`, can you recover it from PowerShell?
9. Write a command that would copy an entire folder and all its contents to a new location.
10. Describe a practical safety check you would perform before running `rm -r` on a folder.
11. What happens if you `cp` a file to a destination where a file with the same name already exists? How would you handle this safely?
12. Compare `mv old_name.txt new_name.txt` vs `mv old_name.txt ~/Documents/new_name.txt`. What is the key difference?
13. Design a workflow to safely delete 50 files matching the pattern `*.bak` from a folder containing 500 files. What commands and verifications would you use?
14. Explain how you could back up all `.scad` files from a project folder into a timestamped backup folder in one command.
15. When organizing a 3D printing project, you need to move completed designs to an archive folder and delete failed prototypes. How would you structure this as a safe, auditable process?

Extension Problems

1. Create a folder tree and copy it to a new location with `cp -r`.
2. Write a one-line command that creates three files named `a b c` and lists them.
3. Move a file into a new folder and confirm the move.
4. Use wildcards to delete files matching a pattern in a safe test folder.
5. Export a listing of the practice folder to `practice_listing.txt`.
6. Create a backup script that copies all `.scad` files from your project folder to a backup folder with timestamp naming.
7. Build a safe deletion workflow: list files matching a pattern, verify count, then delete with confirmation; document the steps.
8. Write a PowerShell script that organizes files by extension into subfolders; test it on a sample folder tree.
9. Create a file operation audit trail: log all copy, move, and delete operations to a text file for review.
10. Develop a project template generator: a script that creates a standard folder structure for a new 3D printing project with essential subfolders.
11. Implement a file conflict handler: write a script that handles cases where

`cp` would overwrite an existing file by renaming the existing file with a timestamp before copying.

12. Create a batch rename operation: use a script to rename all files in a folder from `old_prefix_*` to `new_prefix_*`; test with actual files and verify the results.
13. Build a folder comparison tool: list all files in two folders and identify which files exist in one but not the other; output to a report.
14. Write a destructive operation validator: before executing `rm -r`, create a script that lists exactly what will be deleted, shows file counts by type, and requires explicit user confirmation to proceed.
15. Design a complete project lifecycle workflow: create folders for active projects, completed designs, and archive; include move operations between folders, backup steps, and verification that all files arrive intact.

References

- Microsoft. (2024). *New-Item cmdlet reference*. https://learn.microsoft.com/powershell/module/microsoft.powershell.management/new-item
- Microsoft. (2024). *Copy-Item and Move-Item cmdlets*. https://learn.microsoft.com/powershell/module/microsoft.powershell.management/copy-item
- Microsoft. (2024). *File system operations guide*. https://learn.microsoft.com/powershell/scripting/learn/shell/manipulating-items

Helpful Resources

- New-Item Cmdlet Reference
- Copy-Item and Move-Item
- Remove-Item Cmdlet Reference
- File System Operations Guide
- Safe Deletion Practices

PS-3: Input, Output, and Piping

Duration: 1 class period Prerequisite: PS-2 (File and Folder Manipulation)

Learning Objectives

By the end of this lesson, you will be able to:

- Use `echo` to print text to the screen
- Use `cat` to read file contents
- Use > to redirect output into a file
- Use | (pipe) to send one command's output to another
- Copy output to the clipboard with `clip`
- Open files with a text editor from the command line

Commands Covered

Command	What It Does
`echo "text"`	Print text to the screen
`cat filename`	Print the contents of a file
`> filename`	Redirect output into a file (overwrites)
`>> filename`	Append output to a file (adds to end)
`|`	Pipe - send output from one command to the next
`clip`	Copy piped input to the Windows clipboard
`notepad.exe filename`	Open a file in Notepad

`echo` - Printing Text

`echo` prints text to the screen. It is useful for testing, for writing text into files, and for understanding how piping works.

```
echo "Hello, World"
echo "This is a test"
```

`cat` - Reading Files

`cat` prints the contents of a file to the screen.

```
# Read a text file
cat ~/Documents/notes.txt
# Read an OpenSCAD file
cat ~/Documents/OpenSCAD_Projects/project0.scad
```

With a long file, use `cat filename | more` to read it page by page (press `Space` to advance, `Q` to quit).

> - Redirecting Output to a File

The > symbol redirects output from the screen into a file instead.

```
# Create a file with a single line
echo "Author: Your Name" > header.txt
# Confirm the file was created and has content
cat header.txt
```

Warning

> overwrites the file if it already exists. Use >> to append instead:

```
echo "Date: 2025" >> header.txt
echo "Project: Floor Marker" >> header.txt
cat header.txt
```

| - Piping

The pipe symbol | sends the output of one command to the input of the next. This lets you chain commands together.

```
# List files and send the list to clip (copies to clipboard)
ls -n | clip
# Now paste with Ctrl + V anywhere
```

```
# Search within a file's contents using Select-String (like grep)
cat project0.scad | Select-String "cube"
```

`clip` - Copying to Clipboard

`clip` takes whatever is piped to it and puts it on the Windows clipboard.

```
# Copy your current directory path to the clipboard
pwd | clip
# Copy a file listing to clipboard
ls -n | clip
# Copy the contents of a file to clipboard
cat notes.txt | clip
```

After any of these, press `Ctrl + V` in any application to paste.

Opening Files in Notepad

```
# Open a file in Notepad
notepad.exe ~/Documents/notes.txt
# Open a .scad file
notepad.exe ~/Documents/OpenSCAD_Projects/project0.scad
# Create a new file and open it
notepad new_notes.txt
```

Step-by-step Tasks

1. Create `practice.txt` with three lines using `echo` and >/>>.
2. Read the file with `cat practice.txt`.
3. Pipe the file into `Select-String` to search for a word.
4. Copy the file contents to clipboard with `cat practice.txt | clip`.
5. Redirect `ls -n` into `list.txt` and open it in Notepad. Checkpoints

- After step 3 you should be able to find a keyword using piping.

Quiz - Lesson PS.3

1. What is the difference between > and >>?
2. What does the pipe | do?
3. How do you copy output to the clipboard?
4. How would you page through long output?
5. How do you suppress output to nowhere?

6. True or False: The pipe operator `|` connects the output of one command to the input of another.
7. Explain why redirecting output to a file is useful for screen reader users.
8. Write a command that would search for the word ”sphere” in all `.scad` files in a directory.
9. How would you count the number of lines in a file using PowerShell piping?
10. Describe a practical scenario in 3D printing where you would pipe or redirect command output.
11. What would be the difference in output between `echo "test" > file.txt` (run twice) vs `echo "test" >> file.txt` (run twice)? Show the expected file contents.
12. Design a three-step piping chain: read a file, filter for specific content, and save the results; explain what each pipe does.
13. You have a 500-line `.scad` file and need to find all instances of `sphere()` and count them. Write the command.
14. Explain how `clip` is particularly valuable for screen reader users when working with file paths or long output strings.
15. Describe how you would use pipes and redirection to create a timestamped backup report of all `.stl` files in a 3D printing project folder.

Extension Problems

1. Use piping to count lines in a file (hint: `Select-String -Pattern '.' | Measure-Object`).
2. Save a long `ls -n` output and search it with `Select-String`.
3. Chain multiple pipes to filter and then save results.
4. Practice copying different command outputs to clipboard and pasting.
5. Create a small script that generates a report (counts of files by extension).
6. Build a data processing pipeline: read a CSV file, filter rows, and export results; document each step.
7. Write a script that pipes directory listing to Count occurrences of each file extension; create a summary report.
8. Create a log analysis command: read a log file, filter for errors, and save matching lines to a separate error log.
9. Design a piping workflow for 3D printing file management: find `.stl` files, extract their sizes, and generate a report.
10. Develop a reusable piping function library: create functions for common filtering, sorting, and reporting patterns; test each function with different inputs.
11. Build a complex filter pipeline: read a `.scad` file, extract lines containing specific geometry commands, count each type, and output a summary to both screen and file.
12. Create an interactive piping tool: build a script that accepts user input for a search term, pipes through multiple filters, and displays paginated results.
13. Develop a performance analysis tool: use piping to combine file listing,

metadata extraction, and statistical reporting; export results to a dated report file.
14. Implement a comprehensive error-handling pipeline: read output, catch errors, log them separately, and generate a summary of successes vs failures.
15. Design and execute a real-world project backup workflow: use piping to verify file integrity, count files by type, generate a backup manifest, and create audit logs-all in one integrated command pipeline.

References

- Microsoft. (2024). *Out-File cmdlet for redirection*. https://example.com
- Microsoft. (2024). *Select-String piping reference*. https://example.com
- Microsoft. (2024). *PowerShell pipeline concepts*. https://example.com

Helpful Resources

- Using Out-File for Redirection
- Piping and Select-String
- Get-Content Cmdlet Reference
- Measure-Object for Counting
- PowerShell Pipeline Concept

PS-4: Environment Variables, PATH, and Aliases

Estimated time: 30-45 minutes

Learning Objectives

- Read environment variables with `$env:VARNAME`
- Inspect and verify programs in the `PATH`
- Create temporary aliases and understand making them persistent via the profile

Materials

- PowerShell (with rights to open profile if desired)

Step-by-step Tasks

1. Show your username and home path with `echo $env:USERNAME` and `echo $env:USERPROFILE`.
2. Inspect `echo $env:PATH` and identify whether `openscad` or `code` would be found.
3. Run `Get-Command openscad` and note the result.
4. Create a temporary alias: `Set-Alias -Name preview -Value openscad` and test `preview myfile.scad`.
5. Open your profile (`notepad.exe $PROFILE`) and add the alias line to make it persistent (advanced).

Checkpoints

- After step 3 you can determine whether a program will be found by `PATH`.

Quiz - Lesson PS.4

1. How do you print an environment variable?
2. What is the purpose of `PATH`?
3. How do you check whether `openscad` is available?
4. How do you create a temporary alias?
5. Where would you make an alias permanent?
6. True or False: Environment variables are case-sensitive on all platforms.
7. Explain why having a program in your PATH is useful compared to always using its full file path.
8. Write a command that would create an alias called `slicer` for the OpenSCAD executable.
9. What file would you edit to make an alias persist across PowerShell sessions?
10. Describe a practical benefit of using the `$env:TEMP` directory for temporary files in a 3D printing workflow.
11. You have a custom script at `C:\Scripts\backup_models.ps1` that you want to run from anywhere as `backup-now`. What steps would you take to make this work?
12. Explain the difference between setting an environment variable in the current session vs. adding it to your profile for permanence.
13. Design a profile strategy for managing multiple 3D printing projects, each with different tool paths and directories; show how to structure environment variables for each.
14. If a program is not found by `Get-Command`, what are the possible reasons, and how would you troubleshoot?
15. Describe how you would verify that your PowerShell profile is loading correctly and how to debug issues if aliases or environment variables don't appear after restarting PowerShell.

Extension Problems

1. Add a folder to PATH for a test program (describe steps; do not change system PATH without admin).
2. Create a short profile snippet that sets two aliases and test re-opening PowerShell.
3. Use `Get-Command` to list the path for several common programs.
4. Explore `$env:TEMP` and create a file there.
5. Save a copy of your current PATH to a text file and examine it in your editor.
6. Create a PowerShell profile script that loads custom aliases and environment variables for your 3D printing workflow; test it in a new session.

7. Build a ”project profile” that sets environment variables for CAD, slicing, and print directories; switch between profiles for different projects.
8. Write a script that audits your current environment variables and creates a summary report of what's set and why.
9. Design a custom alias system for common 3D printing commands; document the aliases and their purposes.
10. Create a profile migration guide: document how to export and import your PowerShell profile across machines for consistent workflows.
11. Implement a safe PATH modification script: create a utility that allows you to add/remove directories from PATH for the current session only; show how to make it permanent in your profile.
12. Build a comprehensive profile framework with modules: create separate .ps1 files for aliases, environment variables, and functions; have your main profile load all of them dynamically.
13. Develop an environment validation tool: write a script that checks whether all required programs (OpenSCAD, slicers, etc.) are accessible via PATH; report findings and suggest fixes.
14. Create a project-switching alias system: design a function that changes all environment variables and aliases based on the current project; test switching between multiple projects.
15. Build a profile troubleshooting guide: create a script that exports your current environment state (variables, aliases, PATH) to a timestamped file, allowing you to compare states before and after changes and identify what broke.

References

- Microsoft. (2024). *Environment variables in PowerShell*. https://learn.microsoft.com/powershell/scripting/learn/shell/using-environment-variables
- Microsoft. (2024). *Set-Alias cmdlet reference*. https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/set-alias
- Microsoft. (2024). *Creating and using PowerShell profiles*. https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about_profiles

Helpful Resources

- Environment Variables in PowerShell
- Understanding the PATH Variable
- Set-Alias Cmdlet Reference
- Creating a PowerShell Profile
- Get-Command for Locating Programs

PS-5: Filling in the Gaps - Control Flow, Profiles, and Useful Tricks

Estimated time: 30-45 minutes

Learning Objectives

- Use history and abort commands (`history`, `Ctrl+C`)
- Inspect and edit your PowerShell profile for persistent settings
- Run programs by full path using the `&` operator
- Handle common screen reader edge cases when using the terminal

Materials

- PowerShell and an editor (Notepad/ VS Code)

Step-by-step Tasks

1. Run several simple commands (e.g., `pwd`, `ls -n`, `echo hi`) then run `history` to view them.
2. Use `Invoke-History <n>` to re-run a previous command (replace `<n>` with a history number).
3. Practice aborting a long-running command with `Ctrl + C` (for example, `ping 8.8.8.8`).
4. Open your profile: `notepad.exe $PROFILE`; if it doesn't exist, create it: `ni $PROFILE -Force`.
5. Add a persistent alias line to your profile (example: `Set-Alias -Name preview -Value openscad`), save, and reopen PowerShell to verify.

Checkpoints

- After step 2 you can re-run a recent command by history number.
- After step 5 your alias should persist across sessions.

Quiz - Lesson PS.5

1. How do you view the command history?
2. Which key combination aborts a running command?
3. What does `echo $PROFILE` show?
4. How does the `&` operator help run executables?
5. What is one strategy if terminal output stops being announced by your screen reader?
6. True or False: Using `Ctrl+C` permanently deletes any files created by the command you abort.
7. Explain the difference between `history` and `Get-History` in PowerShell.
8. If you place code in your profile but it doesn't take effect after opening a new PowerShell window, what should you verify?
9. Write a command that would run a program at the path `C:\Program Files\OpenSCAD\openscad.exe` directly.

10. Describe a practical workflow scenario where having keyboard shortcuts (aliases) in your profile would save time.
11. Explain how to re-run the 5th command from your history, and what would happen if that command had file operations (creates/deletes).
12. Design a profile initialization strategy that separates utilities for different projects; explain how you would switch between them.
13. Walk through a troubleshooting workflow: your screen reader stops announcing output after running a long command. What steps would you take to diagnose and resolve the issue?
14. Create a safety checkpoint system: before any destructive operation (mass delete, overwrite), how would you use profile functions and history to verify the command is correct?
15. Develop a comprehensive capstone scenario: integrate everything from PS-0 through PS-5 (navigation, file operations, piping, environment setup, history) to design an automated 3D printing project workflow with error handling and logging.

Extension Problems

1. Add an alias and an environment variable change to your profile and document the behavior after reopening PowerShell.
2. Create a short script that automates creating a project folder and an initial .scad file.
3. Experiment with running OpenSCAD by full path using `&` and by placing it in PATH; compare results.
4. Practice redirecting `Get-Help` output to a file and reading it in Notepad for screen reader clarity.
5. Document three screen reader troubleshooting steps you used and when they helped.
6. Build a comprehensive PowerShell profile that includes aliases, environment variables, and helper functions for your 3D printing workflow.
7. Create a script that troubleshoots common PowerShell issues (module loading, permission errors, command not found); test at least three scenarios.
8. Write a PowerShell function that coordinates multiple tasks: creates a project folder, starts OpenSCAD, and opens slicing software.
9. Design a screen-reader accessibility guide for PowerShell: document commands, outputs, and accessible navigation patterns.
10. Develop an advanced PowerShell workflow: implement error handling, logging, and confirmation prompts for risky operations.
11. Implement a ”undo” system using history: create a function that logs destructive commands (rm, mv, cp -Force) and allows you to review/rollback the last operation.
12. Build a profile debugger: create a script that compares two PowerShell sessions' environment states (variables, aliases, functions) to identify what loaded/failed to load.

13. Develop a multi-project profile manager: design a system where you can switch entire environments (paths, aliases, variables) for different 3D printing projects by running a single command.
14. Create a comprehensive accessibility analyzer: write a script that tests whether key PowerShell commands produce screen-reader-friendly output; document workarounds for commands that don't.
15. Design a complete capstone project: build an integrated automation suite that manages a 3D printing workflow (project setup, file organization, CAD/slicing tool automation, output logging, error recovery, and audit trails) with full error handling and documentation.

References

- Microsoft. (2024). *PowerShell history and recall functionality*. https://learn.microsoft.com/powershell/scripting/learn/shell/using-history
- Microsoft. (2024). *Understanding and creating PowerShell profiles*. https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about_profiles
- Microsoft. (2024). *The call operator (&) for running executables*. https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about_operators#call-operator-

Helpful Resources

- PowerShell History and Recall
- Understanding Profiles
- Invoke-History Cmdlet Reference
- The Call Operator (&)
- Screen Reader Tips and Tricks

PS-6: Advanced Terminal Techniques - Scripts, Functions & Professional Workflows

Duration: 4-4.5 hours (for screen reader users)
Prerequisites: Complete PS-0 through PS-5
Skill Level: Advanced intermediate

This lesson extends PowerShell skills to professional-level workflows. You'll learn to automate complex tasks, write reusable code, and integrate tools for 3D printing workflows.

Learning Objectives

By the end of this lesson, you will be able to:

- Create and run PowerShell scripts (.ps1 files)

- Write functions that accept parameters
- Use loops to repeat tasks automatically
- Automate batch processing of 3D models
- Debug scripts when something goes wrong
- Create professional workflows combining multiple tools

PowerShell Scripts Basics

What's a Script? A script is a file containing PowerShell commands that run in sequence. Instead of typing commands one by one, you put them in a file and run them all at once.

Why use scripts?

- Repeatability: Run the same task 100 times identically
- Documentation: Commands are written down for reference
- Complexity: Combine many commands logically
- Automation: Schedule scripts to run automatically

Creating Your First Script Step 1: Open a text editor

```
notepad.exe my-first-script.ps1
```

Step 2: Type this script

```
# This is a comment - screen readers will read it
Write-Output "Script is running!"
pwd
ls -n
Write-Output "Script is done!"
```

Step 3: Save the file

- In Notepad: Ctrl+S
- Make sure filename ends in `.ps1`
- Save in an easy-to-find location (like Documents)

Step 4: Run the script

```
.\my-first-script.ps1
```

What happens: PowerShell runs each command in sequence and shows output.

Important: Script Execution Policy On some Windows systems, you might get an error about "execution policy". This is a security feature.

To fix it temporarily:

```
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process
```

Then try your script again:

```
.\my-first-script.ps1
```

Note for screen readers: Your screen reader will announce the error if there is one. Use `Get-Help Get-ExecutionPolicy` for more information.

Variables and Parameters

Using Variables Variables store values you want to use later.

Example script:

```
$mypath = "C:\Users\YourName\Documents"
cd $mypath
Write-Output "I am now in:"
pwd
ls -n
```

Breaking it down:

- `$mypath` = variable name (always starts with $)
- `=` = assign the value after this
- `"C:\Users..."` = the value (a path)
- `cd $mypath` = use the variable (replace `$mypath` with its value)

Functions with Parameters A function is reusable code that you can run with different inputs.

Example: A function that lists files in a folder

```
function ListFolder {
    param(
        [string]$path
    )
    Write-Output "Contents of: $path"
    cd $path
    ls -n
}
# Use the function:
ListFolder -path "C:\Users\YourName\Documents"
ListFolder -path "C:\Users\YourName\Downloads"
```

What's happening:

- `function ListFolder` = name of the function
- `param([string]$path)` = the function accepts a parameter called `$path`
- Inside the function, use `$path` like any variable
- Call the function with `-path "value"`

Screen reader tip: When you call a function, PowerShell will announce the results just like any command.

Loops - Repeating Tasks

Loop Over Files Imagine you have 10 SCAD files and want to print their contents. You could do it 10 times manually, or use a loop.

Example: Print every .scad file in a folder

```
$scadFiles = ls -n *.scad
foreach ($file in $scadFiles) {
    Write-Output "=== File: $file ==="
    cat $file
    Write-Output ""
}
```

What's happening:

- `$scadFiles = ls -n *.scad` = find all .scad files and store in variable
- `foreach ($file in $scadFiles)` = for each file, do this:
  - `Write-Output "=== File: $file ==="` = announce which file
  - `cat $file` = show contents
  - `Write-Output ""` = blank line between files

Result: All files printed one after another, organized and readable.

Loop with a Counter Example: Do something 5 times

```
for ($i = 1; $i -le 5; $i++) {
    Write-Output "This is iteration number $i"
    # Do something here
}
```

What's happening:

- `for ($i = 1; $i -le 5; $i++)` = loop from 1 to 5
- `$i` = counter variable (starts at 1, increases each loop)
- `-le` = "less than or equal to" (stop when $i > 5)
- `$i++` = add 1 to $i each time through

Real-World Example - Batch Processing SCAD Files

Scenario You have 10 OpenSCAD (.scad) files in a folder. You want to:

1. List them all
2. Check how many there are
3. For each one, verify it exists

The Script

```
# Batch Processing Script for SCAD Files
$scadFolder = "C:\Users\YourName\Documents\3D_Projects"
$scadFiles = ls $scadFolder -Filter *.scad -Name
Write-Output "Processing SCAD files in: $scadFolder"
Write-Output "Found $($scadFiles.Count) files"
Write-Output ""
foreach ($file in $scadFiles) {
    $fullPath = Join-Path -Path $scadFolder -ChildPath $file
    if (Test-Path -Path $fullPath) {
        Write-Output " Found: $file"
    } else {
        Write-Output " Missing: $file"
    }
}
Write-Output ""
Write-Output "Batch processing complete!"
```

Breaking it down:

- `$scadFolder` = where to look
- `ls $scadFolder -Filter *.scad -Name` = find .scad files, show names only
- `foreach` = process each file
- `Join-Path` = combine folder and filename into full path
- `Test-Path` = check if file exists
- `if` = do different things based on condition

Running the Script

1. Save as `batch-process.ps1`
2. Edit `$scadFolder` to match your real folder
3. Run it:

```
.\batch-process.ps1
```

Screen reader output:

```
Processing SCAD files in: C:\Users\YourName\Documents\3D_Projects
Found 10 files

 Found: model1.scad
 Found: model2.scad
 Found: model3.scad
[... more files ...]
Batch processing complete!
```

Error Handling

Try-Catch Blocks What if something goes wrong? Use try-catch:

Example:

```
try {
    $file = "C:\nonexistent\path\file.txt"
    $content = cat $file
    Write-Output $content
} catch {
    Write-Output "Error: Could not read file"
    Write-Output "Details: $_"
}
```

What's happening:

- `try` = run these commands

- If an error happens, PowerShell jumps to `catch`
- `catch` = handle the error gracefully
- `$_` = the error message

Screen reader advantage: Errors are announced clearly instead of crashing silently.

Validating Input Example: Make sure a folder exists before processing

```
function ProcessFolder {
    param([string]$folderPath)
    if (-not (Test-Path -Path $folderPath)) {
        Write-Output "Error: Folder does not exist: $folderPath"
        return
    }
    Write-Output "Processing folder: $folderPath"
    ls -n $folderPath
}
ProcessFolder -folderPath "C:\Users\YourName\Documents"
```

What's happening:

- `Test-Path` = check if folder exists
- `-not` = if NOT true
- `return` = exit the function early if error

Debugging Scripts

Common Errors and Solutions

Error 1: "Command not found" Cause: Typo in command name

Fix: Check spelling

```
# Wrong:
writ-output "hello"
# Correct:
Write-Output "hello"
```

Error 2: "Variable is null" Cause: Variable was never assigned

Fix: Make sure variable is set before using it

```
$myvar = "hello" # Set first
Write-Output $myvar # Then use
```

Error 3: "Cannot find path" Cause: Wrong folder path

Fix: Verify path exists

```
# Check if path exists:
Test-Path -Path "C:\Users\YourName\Documents"
# If false, the path is wrong
```

Error 4: "Access denied" Cause: Don't have permission

Fix: Run PowerShell as administrator

- Right-click PowerShell -> Run as administrator

Debugging Technique: Trace Output Add `Write-Output` statements to track what's happening:

```
$path = "C:\Users\YourName\Documents"
Write-Output "Starting script. Path is: $path"
$files = ls -n $path
Write-Output "Found $($files.Count) files"
foreach ($file in $files) {
    Write-Output "Processing: $file"
    # Do something with $file
    Write-Output "Done with: $file"
}
Write-Output "Script complete"
```

Your screen reader will announce each step, so you know where errors happen.

Creating Professional Workflows

Example 1: Automated Project Setup Scenario: You start a new 3D printing project regularly. Instead of creating folders manually:

```
function SetupNewProject {
    param([string]$projectName)
    $baseFolder = "C:\Users\YourName\Documents\3D_Projects"
    $projectFolder = Join-Path -Path $baseFolder -ChildPath $projectName
    # Create folder structure
    mkdir $projectFolder -Force
    mkdir "$projectFolder\designs" -Force
    mkdir "$projectFolder\output" -Force
    mkdir "$projectFolder\notes" -Force
    # Create a README
    $readmeContent = @"
# $projectName
Created: $(Get-Date)
## Designs
All .scad files go here.
## Output
STL and other exports go here.
## Notes
Project notes and observations.
"@
    $readmeContent | Out-File -FilePath "$projectFolder\README.txt" -Encoding →
        ↪ utf8
    Write-Output "Project setup complete: $projectFolder"
}
# Use it:
SetupNewProject -projectName "MyKeychain"
SetupNewProject -projectName "PhoneStand"
```

What it does:

- Creates folder structure for a new project
- Sets up subfolders for designs, output, notes
- Creates a README file automatically

Example 2: Batch File Verification Scenario: Before processing, verify all required files exist:

```powershell
function VerifyProjectFiles {
    param([string]$projectFolder)
    $requiredFiles = @(
        "README.txt",
        "designs",
        "output",
        "notes"
    )
    $allGood = $true
    foreach ($item in $requiredFiles) {
        $path = Join-Path -Path $projectFolder -ChildPath $item
        if (Test-Path -Path $path) {
            Write-Output " Found: $item"
        } else {
            Write-Output " Missing: $item"
            $allGood = $false
        }
    }
    if ($allGood) {
        Write-Output "All checks passed!"
        return $true
    } else {
        Write-Output "Some files are missing!"
        return $false
    }
}
# Use it:
$verified = VerifyProjectFiles -projectFolder "C:\Users\YourName\Documents\3D_Projects\MyKeychain"
if ($verified) {
    Write-Output "Safe to proceed with processing"
}
```

Screen Reader Tips for Scripts

Making Script Output Readable Problem: Script runs but output scrolls too fast or is hard to follow

Solution 1: Save to file

```powershell
.\my-script.ps1 > output.txt
notepad.exe output.txt
```

Solution 2: Use Write-Output with clear sections

```powershell
Write-Output "========== STARTING =========="
Write-Output ""
# ... script ...
Write-Output ""
Write-Output "========== COMPLETE =========="
```

Solution 3: Pause between major sections

```powershell
Write-Output "Pausing... Press Enter to continue"
Read-Host
```

Your screen reader will announce the pause, give you time to read output.

Announcing Progress For long-running scripts:

```
$files = ls -n *.scad
$count = 0
foreach ($file in $files) {
    $count++
    Write-Output "Processing $count of $($files.Count): $file"
    # Do something with $file
}
Write-Output "All $count files processed!"
```

Practice Exercises

Exercise 1: Your First Script Goal: Create and run a simple script

Steps:

1. Create file: `notepad.exe hello.ps1`
2. Type:

```
Write-Output "Hello from my first PowerShell script!"
pwd
ls -n
```

3. Save and run: `.\hello.ps1`

Checkpoint: You should see output for each command.

Exercise 2: Script with a Variable Goal: Use a variable to make the script flexible

Steps:

1. Create file: `notepad.exe smart-listing.ps1`
2. Type:

```
$targetFolder = "C:\Users\YourName\Documents"
Write-Output "Listing contents of: $targetFolder"
ls -n $targetFolder
```

3. Edit `$targetFolder` to a real folder on your computer
4. Run: `.\smart-listing.ps1`

Checkpoint: You should see listing of that specific folder.

Exercise 3: Function Goal: Create a reusable function

Steps:

1. Create file: `notepad.exe navigate.ps1`
2. Type:

```
function GoTo {
    param([string]$path)
    if (Test-Path -Path $path) {
        cd $path
        Write-Output "Now in: $(pwd)"
        Write-Output "Contents:"
        ls -n
    } else {
        Write-Output "Path does not exist: $path"
    }
}
# Test the function:
GoTo -path "C:\Users\YourName\Documents"
GoTo -path "C:\Users\YourName\Downloads"
```

3. Run: `.\navigate.ps1`

Checkpoint: Both commands should work, showing contents of each folder.

Exercise 4: Loop Goal: Use a loop to repeat an action

Steps:

1. Create file: `notepad.exe repeat.ps1`
2. Type:

```
Write-Output "Demonstrating a loop:"
for ($i = 1; $i -le 5; $i++) {
    Write-Output "Iteration $i: Hello!"
}
Write-Output "Loop complete!"
```

3. Run: `.\repeat.ps1`

Checkpoint: Should print ”Iteration 1” through ”Iteration 5”.

Exercise 5: Real-World Script Goal: Create a useful script for a real task

Steps:

1. Create a folder: `mkdir C:\Users\YourName\Documents\TestFiles`
2. Create some test files:

```
echo "test" > C:\Users\YourName\Documents\TestFiles\file1.txt
echo "test" > C:\Users\YourName\Documents\TestFiles\file2.txt
echo "test" > C:\Users\YourName\Documents\TestFiles\file3.txt
```

3. Create script: `notepad.exe report.ps1`
4. Type:

```
$folder = "C:\Users\YourName\Documents\TestFiles"
$files = ls -n $folder
Write-Output "=== FILE REPORT ==="
Write-Output "Folder: $folder"
Write-Output "Total files: $($files.Count)"
Write-Output ""
Write-Output "Files:"
foreach ($file in $files) {
    Write-Output " - $file"
}
Write-Output ""
Write-Output "=== END REPORT ==="
```

5. Run: `.\report.ps1`

Checkpoint: Should show report of all files in the test folder.

Quiz - Lesson PS-6

1. What is a PowerShell script?
2. What file extension do PowerShell scripts use?
3. What is a variable and how do you create one?
4. What is a function and why would you use one?
5. How do you run a script?
6. What is a loop and what does `foreach` do?
7. What does `Test-Path` do?
8. How do you handle errors in a script?
9. When would you use `Try-Catch`?
10. What technique makes script output readable for screen readers?

Extension Problems

1. Auto-Backup Script: Create a script that copies all files from one folder to another, announcing progress
2. File Counter: Write a function that counts files by extension (.txt, .scad, .stl, etc.)
3. Folder Cleaner: Script that deletes files older than 30 days (with user confirmation)
4. Project Template: Function that creates a complete project folder structure with all needed files
5. Batch Rename: Script that renames all files in a folder according to a pattern
6. Log Generator: Create a script that records what it does to a log file for later review
7. Scheduled Task: Set up a script to run automatically every day at a specific time
8. File Verifier: Check that all SCAD files in a folder have corresponding STL exports
9. Report Generator: Create a summary report of all projects in a folder
10. Error Tracker: Script that lists all commands that had errors in your recent history

Important Notes

- Always test scripts on small sets of files first before running them on important data
- Save your work regularly - use version control if possible
- Test error handling - make sure errors don't crash silently
- Document your scripts - use comments so you remember what each part does
- Backup before batch operations - if something goes wrong, you have the original

References

- Microsoft PowerShell Scripting Guide: https://docs.microsoft.com/powershell/scripting/
- Function Documentation: https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about_functions_advanced
- Error Handling: https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about_error_handling
- Loops: https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about_foreach

Next Steps: After mastering this lesson, explore PowerShell modules, remoting, and 3D printing integration in the main curriculum.

PowerShell Unit Test - Comprehensive Assessment

Estimated time: 60-90 minutes

Key Learning Outcomes Assessed

By completing this unit test, you will demonstrate:

1. Understanding of file system navigation and path concepts
2. Proficiency with file and folder manipulation commands
3. Ability to redirect and pipe command output
4. Knowledge of environment variables and aliases
5. Screen-reader accessibility best practices in terminal environments
6. Problem-solving and command chaining skills

Target Audience:

Users who have completed PS-0 through PS-5 and need to demonstrate mastery of PowerShell fundamentals.

Instructions:

Complete all sections below. For multiple choice, select the best answer. For short answers, write one to two sentences. For hands-on tasks, capture evidence (screenshots or output files) and submit alongside your answers.

Part A: Multiple Choice Questions (20 questions)

Select the best answer for each question. Each question is worth 1 point.

1. What is the primary purpose of the `PATH` environment variable?
   - A) Store your home directory location
   - B) Tell the shell where to find executable programs

- C) Configure the visual appearance of the terminal
- D) Store the current working directory name

2. Which command prints your current working directory?

   - A) `ls -n`
   - B) `cd`
   - C) `pwd`
   - D) `whoami`

3. What does the ~ symbol represent in PowerShell paths?

   - A) The root directory
   - B) The current directory
   - C) The parent directory
   - D) The home directory

4. How do you list only file names (not full details) in a way that is screen-reader friendly?

   - A) `ls`
   - B) `ls -n`
   - C) `ls -l`
   - D) `cat -n`

5. Which command creates a new empty file?

   - A) `mkdir filename`
   - B) `ni filename`
   - C) `touch filename`
   - D) `echo filename`

6. What is the difference between > and >>?

   - A) > redirects to file, >> displays on screen
   - B) > overwrites a file, >> appends to a file
   - C) They do the same thing
   - D) > is for text, >> is for binary

7. What does the pipe operator `|` do?

   - A) Creates a folder
   - B) Sends the output of one command to the input of another
   - C) Deletes files matching a pattern
   - D) Lists all processes

8. Which command copies a file?

   - A) `mv`
   - B) `rm`
   - C) `cp`
   - D) `cd`

9. How do you rename a file from `oldname.txt` to `newname.txt`?
    - A) `cp oldname.txt newname.txt`
    - B) `mv oldname.txt newname.txt`
    - C) `rename oldname.txt newname.txt`
    - D) `rn oldname.txt newname.txt`
10. What is the purpose of `Select-String`?
    - A) Select files in a directory
    - B) Search for text patterns within a file
    - C) Select a string to copy to clipboard
    - D) Select which shell to use
11. Which key combination allows you to autocomplete a path in PowerShell?
    - A) `Ctrl + A`
    - B) `Ctrl + E`
    - C) `Tab`
    - D) `Space`
12. How do you copy text to the Windows clipboard from PowerShell?
    - A) `cat filename > clipboard`
    - B) `cat filename | clip`
    - C) `copy filename`
    - D) `cat filename | paste`
13. What does `Get-Command openscad` do?
    - A) Opens the OpenSCAD application
    - B) Gets help about the OpenSCAD command
    - C) Locates the full path of the openscad executable
    - D) Lists all available commands
14. Which wildcard matches any single character?
    - A) `*`
    - B) `?`
    - C) `%`
    - D) `#`
15. What is the purpose of the `&` call operator?
    - A) Run a script or executable by full path
    - B) Execute all commands in parallel
    - C) Combine multiple commands
    - D) Create an alias
16. How do you create a temporary alias for a command?
    - A) `alias preview='openscad'`
    - B) `Set-Alias -Name preview -Value openscad`

- C) `New-Alias preview openscad`
- D) `Alias preview = openscad`

17. Where is your PowerShell profile typically stored?
    - A) C:\Program Files\PowerShell\profile.ps1
    - B) The location returned by `echo $PROFILE`
    - C) ~/PowerShell/profile.ps1
    - D) ~/.bashrc
18. How do you abort a long-running command in PowerShell?
    - A) Press `Escape`
    - B) Press `Ctrl + X`
    - C) Press `Ctrl + C`
    - D) Press `Alt + F4`
19. What command shows the history of previously run commands?
    - A) `history`
    - B) `Get-History`
    - C) `Show-History`
    - D) Both A and B
20. How do you permanently set an alias so it persists across PowerShell sessions?
    - A) Use `Set-Alias` in the terminal every time
    - B) Add the `Set-Alias` line to your PowerShell profile
    - C) Use the Windows Control Panel
    - D) Aliases cannot be made permanent

Part B: Short Answer Questions (10 questions)

Answer each question in one to two sentences. Each question is worth 2 points.

1. Explain the difference between absolute and relative paths. Give one example of each.
2. Why is `ls -n` preferred over `ls` for screen reader users? Describe what flag you would use to list only files.
3. What is the purpose of redirecting output to a file, and give an example of when you would use > instead of >>?
4. Describe what would happen if you ran `rm -r ~/Documents/my_folder` and why this command should be used carefully.
5. How would you search for all files with a `.scad` extension in your current directory? Write the command.
6. Explain what happens when you pipe the output of `ls -n` into `clip`. What would you do next?

7. What is an environment variable, and give one example of how you might use it in PowerShell.
8. If a program is not in your `PATH`, what two methods could you use to run it from PowerShell?
9. Describe how you would open a file in Notepad and also add a line to it from PowerShell.
10. What is one strategy you would use if your screen reader stops announcing terminal output while using PowerShell?

Part C: Hands-On Tasks (10 tasks)

Complete each task and capture evidence (screenshots, output files, or command transcripts). Each task is worth 3 points.

Tasks 1-5: File System and Navigation

1. Create a folder structure `~/Documents/PowerShell_Assessment/Projects` using a single command. Capture the `ls -n` output showing the creation.
2. Create five files named `project_1.scad`, `project_2.scad`, `project_3.txt`, `notes_1.txt`, and `notes_2.txt` inside the `Projects` folder. Use wildcards to list only `.scad` files, then capture the output.
3. Copy the entire `Projects` folder to `Projects_Backup` using `cp -r`. Capture the `ls -n` output showing both folders exist.
4. Move (rename) `project_1.scad` to `project_1_final.scad`. Capture the `ls -n` output showing the renamed file.
5. Delete `notes_1.txt` and `notes_2.txt` using a single `rm` command with wildcards. Capture the final `ls -n` output.

Tasks 6-10: Advanced Operations and Scripting

6. Create a file called `my_data.txt` with at least four lines using `echo` and >>. Then read it with `cat my_data.txt` and capture the output.
7. Use `Select-String` to search for a keyword (e.g., "project") in `my_data.txt` and pipe the results to `clip`. Paste the results into Notepad and capture a screenshot.
8. List all files in the `Projects` folder and redirect the output to `projects_list.txt`. Open it in Notepad and capture a screenshot of the file.
9. Create a temporary alias called `myls` that runs `ls -n`, test it, and capture the output. Then explain what would be required to make it permanent.

10. Run `Get-Help Get-ChildItem` and redirect the output to `help_output.txt`. Open the file in Notepad and capture a screenshot showing at least the first page of help content.

Grading Rubric

Section	Questions	Points Each	Total
Multiple Choice	20	1	20
Short Answer	10	2	20
Hands-On Tasks	10	3	30
Total	40	-	70

Passing Score: 49 points (70%)

Helpful Resources for Review

- PowerShell Command Reference
- Navigation and File System
- Using Pipes and Filtering
- Profile and Aliases
- Screen Reader Accessibility Tips

Submission Checklist

- ☐ All 20 multiple choice questions answered
- ☐ All 10 short answer questions answered (1-2 sentences each)
- ☐ All 10 hands-on tasks completed with evidence captured
- ☐ Files/screenshots organized and labeled clearly
- ☐ Submission includes this checklist

Screen Reader Accessibility Guide for PowerShell

Target Users: NVDA, JAWS, and other screen reader users
Last Updated: 2026

This guide supports the PowerShell Foundation curriculum and helps screen reader users navigate and work efficiently with PowerShell on Windows.

Table of Contents

5. Working with Long Output
6. Keyboard Shortcuts Reference
7. Troubleshooting

Getting Started with Screen Readers

Which Screen Reader Should I Use? NVDA is free and often recommended for new users; JAWS is a powerful commercial option. Also consider Dolphin SuperNova and Windows Narrator:

- Dolphin SuperNova: commercial speech, braille and magnification (check vendor docs for keyboard mappings).
- Windows Narrator: built into Windows and useful for quick access without installing third-party software.

Before You Start

1. Start your screen reader before opening PowerShell.
2. Open PowerShell and listen for the window title and prompt.
3. If silent, press Alt+Tab to find the window.

What is PowerShell? PowerShell is the modern Windows shell and scripting environment. Common commands include `Get-ChildItem`, `Get-Content`, and `Out-File`. PowerShell provides richer objects and piping than CMD.

NVDA-Specific Tips

NVDA is available from https://www.nvaccess.org/

Dolphin SuperNova Dolphin SuperNova: https://yourdolphin.com/supernova/ — commercial option providing speech and magnification; consult vendor guides for features and commands.

Windows Narrator Windows Narrator: https://support.microsoft.com/narrator — built-in Narrator has a different set of commands; it can be enabled via Windows Settings > Accessibility.

Key Commands for PowerShell

Command	What It Does
NVDA+Home	Read the current line (your command or output)
NVDA+Down Arrow	Read from cursor to end of screen
NVDA+Up Arrow	Read from top to cursor

Command	What It Does
NVDA+Page Down	Read next page
NVDA+Page Up	Read previous page
NVDA+F7	Open the Review Mode viewer (can scroll through text)

Example: Reading Long Output If `Get-ChildItem` produces many lines, redirect to a file and open it in Notepad:

```
Get-ChildItem -Name > list.txt
notepad list.txt
```

`-Name` prints one item per line (screen reader friendly).

JAWS-Specific Tips

JAWS is available from https://www.freedomscientific.com/

Key Commands for PowerShell

Command	What It Does
Insert+Down Arrow	Read line by line downward
Insert+Up Arrow	Read line by line upward
Insert+Page Down	Read next page of text
Insert+Page Up	Read previous page of text

Example: Reading Long Output

1. Redirect: `Get-ChildItem -Name > list.txt`
2. Open Notepad: `notepad list.txt`
3. Use Insert+Ctrl+Down to read full contents.

General Terminal Accessibility

Understanding the PowerShell Layout PowerShell shows a title bar, content area, and a prompt that looks like:

```
PS C:\Users\YourName>
```

Navigation Sequence

1. Screen reader announces the title
2. Then it announces the prompt line
3. Anything above prompt is prior output

Working with Long Output

Solution 1: Redirect to a File

```
Get-ChildItem -Name > list.txt
notepad list.txt
```

Solution 2: Use Pagination

```
Get-Content largefile.txt | more
```

Solution 3: Filter Output

```
Get-ChildItem -Name | Where-Object { $_ -like "*.scad" }
```

Solution 4: Count Before Displaying

```
(Get-ChildItem -Name).Count
```

Keyboard Shortcuts Reference

Key	Action
Up Arrow	Show previous command
Down Arrow	Show next command
Tab	Auto-complete file/folder names
Home	Jump to start of line
End	Jump to end of line
Ctrl+C	Stop command
Enter	Run command

Troubleshooting

Problem: "I Can't Hear the Output"

1. Redirect to file and open in Notepad.
2. Use End to jump to the end of text.

Problem: "Tab Completion Isn't Working"

1. Type some characters before Tab.

Problem: "Command Not Found"

1. Use `Get-Command programname` to check availability.

Pro Tips

1. Use `Get-ChildItem -Name` for one-per-line listings.
2. Use `Out-File -FilePath list.txt` to capture output with encoding options.

Recommended Workflow

1. `Set-Location` (or `cd`) to the project folder
2. `Get-ChildItem -Name` to list files
3. Redirect large output to files and open in Notepad

Additional Resources

- NVDA Documentation: https://www.nvaccess.org/documentation/
- JAWS Documentation: https://www.freedomscientific.com/support/
- PowerShell Documentation: https://docs.microsoft.com/powershell
- NVDA Documentation: https://www.nvaccess.org/documentation/
- JAWS Documentation: https://www.freedomscientific.com/support/
- Dolphin SuperNova: https://yourdolphin.com/supernova/
- Windows Narrator: https://support.microsoft.com/narrator

Windows Command Line

This section covers terminal fundamentals, screen reader accessibility, and command-line basics needed before diving into 3D design with OpenSCAD. This is an alternate pathway to the PowerShell curriculum, using Windows Command Prompt (CMD) instead.

Time commitment: ~10 hours
Skills gained: Terminal navigation, file operations, basic scripting, keyboard-only workflow mastery

Note

CMD is simpler than PowerShell but has fewer advanced features. Both pathways teach the same fundamental concepts.

Windows Command Prompt (CMD) for Screen Reader Users - Complete Curriculum Overview

Welcome! This curriculum teaches you how to use Windows Command Prompt (CMD) as a screen reader user, starting from zero experience and building to practical skill levels.

Last Updated: February 2026
Total Duration: 30-45 hours of instruction and practice (for screen reader users)
Target Users: Screen reader users — NVDA, JAWS, Windows Narrator, and Dolphin SuperNova are all covered throughout this curriculum.

Note: Time estimates reflect the additional time needed for screen reader navigation, text-to-speech processing, and careful keyboard-based workflows.

Alternate to: PowerShell curriculum (same concepts, different command syntax)

Why Learn Windows Command Prompt (CMD)?

For Everyone

- **Speed:** Text commands are often faster than clicking through menus.
- **Simplicity:** CMD has fewer commands than PowerShell, but they are straightforward.
- **Accessibility:** CMD works reliably with all major screen readers.
- **Precision:** Exact control over what your computer does.

For 3D Printing (Our Focus)

- **Batch Operations:** Process multiple 3D models at once.
- **Accessibility:** Many 3D design tools are scriptable from CMD.
- **Reproducibility:** Same settings, every time.
- **Integration:** Connect OpenSCAD and other tools together.

For Screen Reader Users Specifically

- **Great Accessibility:** CMD works well with NVDA, JAWS, Windows Narrator, and Dolphin SuperNova.
- **No Mouse Needed:** Everything is keyboard-based.
- **Text-Based:** Output is naturally readable by screen readers.
- **Stability:** Unlike GUIs, terminal interactions are consistent.
- **Simpler Syntax:** CMD commands are more straightforward than PowerShell for beginners.

Curriculum Structure

Phase 1: Absolute Beginner → Comfortable User

Lesson	Duration	What You Will Learn
CMD-Pre: Your First Terminal	1.5-2 hours	Opening CMD, first commands, basic navigation
CMD-0: Getting Started	1.5 hours	Paths, shortcuts, command basics
CMD-1: Navigation	2-2.5 hours	Moving around the file system confidently

Goal: You can navigate to any folder and see what is in it with your screen reader.

Phase 2: Intermediate User □ Power User

Lesson	Duration	What You Will Learn
CMD-2: File & Folder Manipulation	2.5-3 hours	Create, copy, move, delete files/folders
CMD-3: Input, Output & Redirection	2.5-3 hours	Redirect output, pipe commands
CMD-4: Environment Variables & Shortcuts	2-2.5 hours	Automate settings, create shortcuts
CMD-5: Filling in the Gaps	2-2.5 hours	Batch files, history, debugging

Goal: You can create folders, manage files, and combine commands to accomplish complex tasks.

Phase 3: Professional Skills

Lesson	Duration	What You Will Learn
CMD-6: Advanced Techniques	2.5-3 hours	Scripting, loops, automation workflows
CMD Unit Test	2.5-4 hours	Comprehensive assessment

How to Use This Curriculum

If You Have Never Used a Terminal Before   Start here and go in order:

1. Do **CMD-Pre: Your First Terminal**.
2. Continue with CMD-0, CMD-1, and so on.

Do not skip steps — each lesson builds on the previous one.

If You Have Used a Terminal Before (But Not with a Screen Reader)

1. Quickly review **CMD-Pre** (it has the screen reader focus you need).
2. Move to **CMD-0** for deeper path understanding.

If You Are Experienced with Terminal and Screen Readers

1. Jump to the specific lesson you need (CMD-2, CMD-3, etc.).
2. Use the **Quick Reference** sections.
3. Skip exercises and take the quizzes to verify knowledge.

How Each Lesson Is Structured

Every lesson contains:

1.

   Learning Objectives — What you will be able to do after the lesson

2. **Key Commands** — The important ones to memorize.
3. **Step-by-Step Examples** — How to actually do it.
4. **Practice Exercises** — Hands-on work (critical: do not skip these).
5. **Quiz Questions** — Check your understanding.
6. **Extension Problems** — Go deeper if interested.

How to get through each lesson:

1. Read the learning objectives.
2. Do the step-by-step examples alongside (type the commands yourself).
3. Complete the practice exercises.
4. Take the quiz honestly.
5. Try extension problems if you have time.
6. Move to the next lesson when you can answer the quiz questions confidently.

Estimated time: 2-3 hours per lesson for screen reader users, depending on practice time.

Screen Reader Tips Throughout the Curriculum

All four screen readers are covered in each lesson:

- **NVDA** — Free, excellent terminal support. Available at https://example.com
- **JAWS** — Commercial, powerful, widely used in workplaces. Available at https://example.com
- **Windows Narrator** — Free, built into Windows, minimal setup required.
- **Dolphin SuperNova** — Commercial, with optional magnification. Available at https://example.com

If your screen reader is not listed in a specific tip, refer to the **Screen Reader Accessibility Guide** for equivalent commands across all four readers.

Quick Start Guide (First 45-60 Minutes)

If you have 45-60 minutes right now:

1. Open Command Prompt (Windows key □ type `cmd` □ Enter).
2. Run these commands:

```
cd
dir /B
cd Documents
cd
```

3. Notice how your screen reader reads each output.
4. Create a file:

```
echo I am learning CMD > learning.txt
type learning.txt
```

That is the core experience. Now read CMD-Pre for the full explanation.

Common Questions Before Starting

Q: PowerShell or CMD? Which should I learn? Both teach the same fundamental concepts. CMD has simpler syntax and is better for beginners. PowerShell is more powerful and better for advanced automation. This curriculum is CMD; there is a separate PowerShell curriculum if you want that pathway. **Choose one and stick with it.**

Q: What if I use a screen reader not listed here? The fundamentals work the same across all screen readers. Check your screen reader's documentation for the equivalent of these functions: "read current line," "read from here to end," and "move through output line by line." All screen readers have these capabilities.

Q: I am intimidated. Is this really for me? Yes. This curriculum is designed specifically for people with no terminal experience AND who use screen readers. You start with absolute basics and build up. There is nothing to be afraid of — the terminal cannot hurt your computer just by navigating and listing files.

Q: How long will this take? Realistically:

- Minimum (lessons only, no exercises): 15-18 hours.
- Normal (lessons and exercises): 30-45 hours.
- With extension problems: 45-60 hours or more.

Spread it over weeks or months. Go at your pace.

Q: What if I forget things? That is normal. Solutions:

1. Come back to this overview page.
2. Jump back to the relevant lesson for a quick review.
3. Use the quiz questions to self-test.
4. Check the Screen Reader Accessibility Guide for troubleshooting.

Important Rules

Rule 1: Always Know Where You Are Every session, first thing:

```
cd
```

If you do not know your path, you will get lost. Do not move until you know where you are.

Rule 2: Check Before You Delete Before deleting anything:

```
dir /B
```

Make sure you are deleting the right thing. Deleted files may not be recoverable from the command line.

Rule 3: Use `dir /B` for Listings The `/B` flag gives a clean, one-per-line listing that is easy for all screen readers to follow:

```
dir /B
```

Rule 4: When Output Is Confusing, Redirect to a File

```
dir /B > output.txt
notepad.exe output.txt
```

This is always clearer for all screen readers than reading terminal output directly.

Rule 5: Save Everything You Create Create a practice folder and keep everything there:

```
mkdir my-practice-folder
cd my-practice-folder
```

Troubleshooting: "Nothing Works!"

1. **Cannot hear CMD at all?**
   - Make sure your screen reader is running BEFORE opening CMD.

    - Try **Alt+Tab** to cycle to the CMD window.
    - Restart both your screen reader and CMD.

2. **Commands not working?**

    - Check spelling carefully.
    - Make sure you pressed Enter.
    - Try `help` for a list of available commands.
    - Try `command /?` (for example, `dir /?`) for help on a specific command.

3. **Cannot read the output?**

    - Redirect to file: `command > output.txt`
    - Open in Notepad: `notepad.exe output.txt`
    - This works with all four screen readers.

4. **Something ran forever?**

    - Press **Ctrl+C** to stop it.

5. **JAWS stopped working?**

    - If using JAWS in demo mode, the session limit is approximately 40 minutes. Restart your computer to reset the demo.

6. **Completely confused?**

    - Go back to CMD-Pre and start over.
    - Work through every single exercise slowly.
    - Ask an instructor or peer for help.

Resources

Official Documentation

- **Windows CMD Reference:** https://example.com
- **Microsoft Learn:** https://example.com

Screen Reader Guides

- **NVDA:** https://example.com
- **JAWS:** https://example.com
- **Windows Narrator:** https://example.com
- **Dolphin SuperNova:** https://example.com

Curriculum Map

```
START HERE
    |
CMD-Pre: Your First Terminal (absolute beginner entry point)
    |
CMD-0: Getting Started (paths and navigation foundations)
    |
CMD-1: Navigation (comfortable moving around)
    |
CMD-2: File & Folder (create, move, delete)
    |
CMD-3: Input & Output (redirect commands)
    |
CMD-4: Variables & Shortcuts (automation foundations)
    |
CMD-5: Filling in the Gaps (batch files and history)
    |
CMD-6: Advanced Techniques (scripts, loops, workflows)
    |
CMD Unit Test (comprehensive assessment)
    |
NEXT: 3D Printing Integration Lessons
```

Questions? Feedback? Stuck? Refer back to this page and try the Troubleshooting section. All the tools you need are here.

Now open CMD-Pre and get started!

CMD-Pre: Your First Terminal - Screen Reader Navigation Fundamentals

Duration: 1.5-2 hours (for screen reader users)
Prerequisites: None — this is the starting point
Accessibility Note: This lesson is designed specifically for screen reader users. Tips are provided for NVDA, JAWS, Windows Narrator, and Dolphin SuperNova.

What is a Terminal?

A terminal (also called command line or Command Prompt) is a text-based interface where you type commands instead of clicking buttons. Think of it like sending written instructions to your computer instead of pointing and clicking.

Why learn this?

- Faster and more precise work, especially for 3D printing scripts and automation.
- Essential for using tools like OpenSCAD.
- Accessibility: command-line tools work reliably with screen readers — output is plain text.
- Simple automation without complex graphical interfaces.

Opening Command Prompt for the First Time

Method 1: Search (Easiest)

1. Press the **Windows key** alone.
2. Type: `cmd`
3. You will hear search results appear.
4. Press **Enter** to open the first result (Command Prompt).

Method 2: Using the Run Dialog

1. Press **Windows key + R** (opens the Run dialog).
2. Type: `cmd`
3. Press **Enter**.

Method 3: From the Start Menu

1. Press the **Windows key**.
2. Navigate to **Windows System** or **Windows Tools** (depending on your Windows version).
3. Open **Command Prompt**.

First Connection: Understanding the Prompt When Command Prompt opens, your screen reader will announce the window title and then the **prompt**. The prompt is where you type commands.

What you will hear:

```
C:\Users\YourName>
```

What this means:

- `C:\Users\YourName` = Your current location (the path).
- > = The prompt is ready for input.

Your cursor is right after the >. This is where you type.

Your First Commands (Screen Reader Edition)

Command 1: ”Where Am I?” — `cd` Running `cd` with no arguments shows your current directory.

Type this and press Enter:

```
cd
```

What you will hear: Your screen reader announces the current path, for example:

```
C:\Users\YourName
```

Understanding paths:

- A path shows your location in the file system, like a mailing address.
- Windows paths use backslashes: `C:\Users\YourName\Documents`
- Think of it as nested folders: `C:\` (the main drive) □ `Users` □ `YourName` □ `Documents`

Command 2: "What's Here?" — `dir` **Type this and press Enter:**

```
dir /B
```

What you will hear: Your screen reader announces folder and file names, one per line. The `/B` flag gives a clean "bare" listing — names only, no dates or sizes — which is much easier to follow with a screen reader.

If you want more detail (sizes, dates), run `dir` without `/B`. But for navigation, `dir /B` is preferred.

Command 3: "Go There" — `cd Documents` **Type this and press Enter:**

```
cd Documents
```

What you will hear: The prompt changes to show your new location:

```
C:\Users\YourName\Documents>
```

Practice navigation:

1. Run `cd` to confirm you are in Documents.
2. Run `dir /B` to see what is there.
3. Go back up: `cd ..` (the `..` means "go up one level").
4. Run `cd` again to confirm.
5. Return to Documents: `cd Documents`

Reading Screen Reader Output (Critical Skills)

Dealing with Long Lists When you run `dir` in a folder with many files, the list may be very long. Here is how to manage it:

Solution 1: Save to a File

```
dir /B > list.txt
notepad.exe list.txt
```

This saves the listing to a file and opens it in Notepad, where you can read it calmly with your screen reader.

Solution 2: Use Pause

```
dir /B | more
```

Shows output one page at a time. Press **Space** to go to the next page, **Q** to quit. Note: some screen readers read `more` output less reliably than a saved file — if this is difficult, use Solution 1.

Tab Completion Tab completion is one of the most powerful screen reader techniques in the terminal.

How it works:

1. Type the first few letters of a folder or file name.
2. Press **Tab**.
3. Command Prompt automatically completes the rest.

Example:

1. You are at `C:\Users\YourName>`
2. Type: `cd Doc`
3. Press **Tab**
4. Command Prompt completes it to: `cd Documents`
5. Press **Enter** to navigate there.

With a screen reader, when you press Tab the screen reader announces the completed command — much faster and more accurate than typing the whole thing.

Creating and Viewing Files

Create a Simple File

```
echo Hello, Command Prompt! > hello.txt
```

- `echo` outputs text.
- > redirects it to a file called `hello.txt`.

Read the File Back

```
type hello.txt
```

Your screen reader announces the file contents.

Open and Edit the File

```
notepad.exe hello.txt
```

Opens the file in Notepad for comfortable editing with your screen reader.

Essential Keyboard Shortcuts

Key Combination	What It Does
Up Arrow	Shows your previous command (press again for earlier commands)
Down Arrow	Shows a more recent command in history
Tab	Auto-completes folder and file names
Ctrl+C	Stops a running command
cls	Clears the screen (type this and press Enter)
Enter	Runs the command you have typed
Home	Moves cursor to start of command line
End	Moves cursor to end of command line

Screen Reader-Specific Tips

NVDA Users NVDA is available free from https://example.com

1. **After running a command**, press **NVDA+Home** (Insert+Home or CapsLock+Home) to read the current line.
2. **To read all output**, press **NVDA+Down Arrow** to read from the cursor to the end of the screen.
3. **To browse previous output**, use the NVDA review cursor: press **Numpad 8/2** (desktop) or the equivalent laptop keys to move up and down through lines.
4. **If output scrolled past**, redirect to a file: `command > output.txt`, then `notepad.exe output.txt`.
5. **To open NVDA settings**: press **NVDA+N** (opens NVDA menu), then go to Preferences.

   NVDA key note: The NVDA modifier key is **Insert** by default on desktops. On laptops it is often **CapsLock**. Check your NVDA settings if these shortcuts do not work as described.

JAWS Users JAWS is available from https://example.com

1. **After running a command**, press **Insert+Up Arrow** to re-read the current line.
2. **To read all output**, press **Insert+Ctrl+Down** to read to the end of the screen.
3. **To move through output line by line**, press **Insert+Down Arrow** repeatedly.

4. **If output is too long**, redirect to a file: `command > output.txt`, then `notepad.exe output.txt`.
5. **JAWS demo mode reminder:** If you are using JAWS without a license, sessions are limited to approximately 40 minutes. Save your work and restart the session if JAWS stops responding.

   JAWS key note: The JAWS modifier key is **Insert** by default. On some laptops it can be set to **CapsLock** in JAWS Settings Center.

Windows Narrator Users Windows Narrator is built into Windows 10 and Windows 11. Enable it with **Windows+Ctrl+Enter**.

1. **After running a command**, press **Narrator+D** (CapsLock+D or Insert+D) to read the current line.
2. **To read all output from here**, press **Narrator+R** to read from the cursor position.
3. **To move through output**, use **Narrator+Up/Down Arrow** in scan mode (press **Narrator+Space** to toggle scan mode on).
4. **For long outputs**, always redirect to a file: `command > output.txt`, then `notepad.exe output.txt`. Notepad is more comfortable for Narrator users than reading directly from the terminal.
5. **Narrator settings**: Windows key □ Settings □ Accessibility □ Narrator.

   Narrator key note: The Narrator modifier key is **CapsLock** or **Insert**. You can change it in Settings □ Accessibility □ Narrator □ Keyboard shortcuts.

Dolphin SuperNova Users Dolphin SuperNova is available from https://example.com

1. **After running a command**, press **CapsLock+L** to read the current line.
2. **To read all output**, press **CapsLock+Numpad Plus** (say all from current position).
3. **To move through output**, press **CapsLock+Numpad 8** (up) and **CapsLock+Numpad 2** (down) to move line by line through the review buffer.
4. **For long outputs**, redirect to a file: `command > output.txt`, then `notepad.exe output.txt`.
5. **SuperNova settings**: press **CapsLock+SpaceBar** to open the SuperNova Control Panel.

   Dolphin key note: The SuperNova modifier key is **CapsLock** by default. This can be changed in the SuperNova Control Panel □ Keyboard.

Common Issue: "I Can't Hear the Output"

This happens to everyone starting out. Here are the solutions:

1. **Make sure your screen reader was running before you opened Command Prompt.** If not, close Command Prompt, confirm your screen reader is running, then reopen it.
2. **Press End** to make sure your cursor is at the prompt.
3. **Redirect to a file**: `command > output.txt`, then `notepad.exe output.txt`. This always works with all screen readers.
4. **Use the screen reader's review commands** listed above to scan back through previous output.

Practice Exercises

Complete these in order. Take your time.

Exercise 1: Basic Navigation

1. Open Command Prompt.
2. Run `cd` and note your location.
3. Run `dir /B` and listen to the listing.
4. Type `cd Documents` and press Enter.
5. Run `cd` to confirm your new location.
6. Run `dir /B` in this new location.

Goal: You know where you are and what is around you.

Exercise 2: Using Tab Completion

1. In your home directory, type `cd D` (just those two characters).
2. Press **Tab**.
3. Command Prompt should auto-complete to a folder starting with D.
4. Repeat with other folder names.
5. Try `cd Down` and Tab to `Downloads`.

Goal: Tab completion feels natural.

Exercise 3: Creating and Viewing Files

1. Create a file: `echo Test content > test.txt`
2. View it: `type test.txt`
3. Create another: `echo Line 2 > another.txt`
4. List both: `dir /B *.txt`

Goal: You understand create, view, and list operations.

Exercise 4: Going Up Levels

1. Navigate into several folders: `cd Documents`, then `cd folder1`, etc.
2. From inside, run `cd ..` multiple times to come back up.
3. After each `cd ..`, run `cd` to confirm your location.

Goal: You are comfortable with relative navigation using `..`

Exercise 5: Redirecting Output

1. Create a listing: `dir /B > directorylist.txt`
2. Open it: `notepad.exe directorylist.txt`
3. Read it with your screen reader.
4. Close Notepad.

Goal: You can save long outputs to files for easier reading.

Checkpoint Questions

After this lesson, you should be able to answer all ten with confidence:

1. What does `cd` do with no arguments?
2. What does `dir /B` do?
3. Why do we use `/B` with `dir`?
4. What is your current path right now?
5. How do you navigate to a new folder?
6. How do you go up one level?
7. What is the Tab key for?
8. What does `echo text > file.txt` do?
9. How do you read a file back in the terminal?
10. How do you stop a command that is running?

Answer all 10 before moving to CMD-0.

Common Questions

Q: Should I use Command Prompt or PowerShell?
A: Command Prompt is simpler and a good starting point. PowerShell is more powerful. Both work well with all four screen readers covered in this curriculum. Start with whichever your curriculum pathway specifies.

Q: Why is my screen reader not reading the output?
A: This is common. Use `command > file.txt` to save output to a file, then open it with `notepad.exe file.txt` for reliable reading.

Q: What if I type something wrong?
A: Press **Enter** and you will see an error message. Type the correct command on the next line. No harm done.

Q: How do I get help with a command?
A: Type `help command-name` (for example, `help cd`). Or type `command /?` for built-in help (for example, `dir /?`).

Next Steps

Once comfortable with these basics, move to **CMD-0: Getting Started** for deeper path understanding, then continue through CMD-1 to CMD-5 for full terminal mastery.

Troubleshooting

Issue	Solution
Command Prompt won't open	Press Windows+R, type `cmd`, press Enter
Cannot hear the output	Redirect to a file: `command > output.txt`, then `notepad.exe output.txt`
Tab completion not working	Type at least one character before pressing Tab
Command not found	Check spelling; try `help` for a list of available commands
Stuck in a command	Press **Ctrl+C** to stop it
JAWS stops working mid-session	If using JAWS demo: restart computer, demo sessions last ~40 minutes

---

Resources

- **NVDA:** https://example.com
- **JAWS:** https://example.com
- **Windows Narrator guide:** https://example.com
- **Dolphin SuperNova:** https://example.com
- **Windows CMD Reference:** https://example.com

CMD-0: Getting Started - Layout, Paths, and the Shell

Estimated time: 20-30 minutes

Learning Objectives

- Launch Command Prompt and locate the prompt
- Understand path notation and shortcuts (`C:\`, `..`, `.`)
- Use tab completion to navigate quickly

Materials

- Computer with Windows
- Editor (Notepad/VS Code)

Step-by-step Tasks

1. Open Command Prompt and note the prompt (it includes the current path).
2. Run `cd` and say or note the printed path.
3. Use `dir /B` to list names in your home directory.
4. Practice `cd Documents`, `cd ..` and `cd \` until comfortable.
5. Try tab-completion: type `cd D` and press Tab.

Checkpoints

- Confirm you can state your current path and move to `Documents`.

Quiz - Lesson CMD.0

1. What is a path?
2. What does `..` mean?
3. How do you autocomplete a path?
4. How do you go up one directory?
5. What command lists only names (`dir` flag)?
6. True or False: On Windows, CMD uses backslashes (\) in paths, but forward slashes (/) are also accepted in some contexts.
7. Explain the difference between an absolute path and a relative path.
8. If you are in `C:\Users\YourName\Documents` and you type `cd ..`, where do you end up?
9. What happens when you press Tab while typing a folder name in Command Prompt?
10. Describe a practical reason why understanding paths is important for a 3D printing workflow.
11. What does `C:\` mean in a path, and when would you use it?
12. If a folder path contains spaces (e.g., `Program Files`), how do you navigate to it with `cd`?
13. Explain what the prompt `C:\Users\YourName>` tells you about your current state.
14. How would you navigate to your home directory from any location using a single command?

15. What is the advantage of using relative paths (like `..`) versus absolute paths in automation scripts?

Extension Problems

1. Create a nested folder and practice `cd` into it by typing partial names and using Tab.
2. Use `dir /B /A:F` to list only files in a folder.
3. Save `cd` output (the path) to a file and open it in Notepad.
4. Try `cd` into a folder whose name contains spaces; observe how quotes are handled.
5. Create a short note file and open it from Command Prompt.
6. Build a folder structure that mirrors your project organization; navigate to each level and document the path.
7. Use `echo %cd%` to print your current path and save it to a file.
8. Investigate special paths (e.g., `%USERPROFILE%`, `%TEMP%`); write down what each contains and when you'd use them.
9. Compare absolute vs. relative paths by navigating to the same folder using each method; explain which is easier for automation.
10. Create a batch file that changes to a frequently-used folder and lists its contents in one command; test it from different starting locations.
11. Navigate to three different locations and at each one note the prompt, the path from `cd`, and verify you understand what each shows.
12. Create a complex folder tree (at least 5 levels deep) and navigate it using only relative paths; verify your location at each step.
13. Document all path shortcuts you know (`C:\`, `..`, `.`) and demonstrate each one works as expected.
14. Write a guide for a peer on how to understand the Command Prompt and path notation without using GUI file explorer.
15. Create a troubleshooting flowchart: if someone says "I don't know where I am," what commands do you give them to find out?

References

- Microsoft. (2024). *Command line reference*. https://example.com
- Microsoft. (2024). *Using the cd command*. https://example.com

Helpful Resources

- Windows Command Reference
- CD Command Guide
- DIR Command Reference
- Understanding Windows Paths

CMD-1: Navigation - Moving Around Your File System

Duration: 1.5-2 hours (for screen reader users)
Prerequisites: CMD-Pre and CMD-0
Learning Objectives:

- Navigate confidently to any folder location
- Understand and use relative versus absolute paths
- Use shortcuts to move between common folders
- Organize your file system logically

Core Navigation Commands

Command: `cd` (Change Directory)

```
cd FolderName :: Go into a folder
cd .. :: Go up one level
cd \ :: Go to root of current drive (e.g., C:\)
cd %USERPROFILE% :: Go to home directory
```

Command: `dir /B` (List Contents)

```
dir /B :: List all names (one per line, screen-reader friendly)
dir /B /A:D :: List only directories/folders
dir /B /A:F :: List only files
```

Command: Show Current Location

```
cd :: Running cd with no arguments shows current path
```

Understanding Paths

Absolute Paths (Full Address) An absolute path starts from the root and shows the complete location:

```
C:\Users\YourName\Documents\3DPrinting
```

- `C:\` — The C drive root
- `Users` — First folder
- `YourName` — Your user folder
- `Documents` — Documents folder
- `3DPrinting` — Your 3D printing folder

Navigate there directly:

```
cd C:\Users\YourName\Documents\3DPrinting
```

Relative Paths (Directions From Here)

```
cd Documents\3DPrinting :: Go into Documents, then into 3DPrinting
cd .. :: Go up one level
cd ..\.. :: Go up two levels
```

Shortcuts:

- `.` = Current folder
- `..` = Parent folder (up one level)
- `%USERPROFILE%` = Your home folder

Common Navigation Patterns

Pattern 1: Navigating Down

```
C:\>cd Users
C:\Users>cd YourName
C:\Users\YourName>cd Documents
C:\Users\YourName\Documents>
```

Pattern 2: Navigating Up

```
C:\Users\YourName\Documents>cd ..
C:\Users\YourName>cd ..
C:\Users>cd ..
C:\>
```

Pattern 3: Jump to a Known Path

```
C:\Deep\Folder\Somewhere>cd C:\Users\YourName
C:\Users\YourName>
```

Pattern 4: Going Home

```
cd %USERPROFILE%
```

Special Folders and Shortcuts

```
%USERPROFILE% :: Your home folder
%USERPROFILE%\Desktop :: Your Desktop
%USERPROFILE%\Documents :: Your Documents
%USERPROFILE%\Downloads :: Your Downloads
%ProgramFiles% :: Program Files folder
C:\ :: Root of C drive
```

Tab Completion (Essential Skill)

1. Type the first few characters of a folder name.
2. Press **Tab**.
3. Command Prompt completes it.

Example:

```
C:\Users\YourName>cd Doc [press Tab]
C:\Users\YourName>cd Documents
```

- Your screen reader announces the completed name immediately.
- Press **Tab** multiple times to cycle through all matches.

Screen Reader Tips for Navigation

NVDA Users

1. After each `cd` command, run `cd` alone to confirm location. Press **NVDA+Home** (Insert+Home) to read the current line.
2. Press **NVDA+Down Arrow** to read through `dir /B` output.
3. For long listings: `dir /B > listing.txt`, then `notepad.exe listing.txt`.

JAWS Users

1. After each `cd` command, press **Insert+Up Arrow** to re-read the current line.
2. Press **Insert+Down Arrow** repeatedly to read output line by line.
3. For long listings: `dir /B > listing.txt`, then `notepad.exe listing.txt`.

Windows Narrator Users

1. After each `cd` command, press **Narrator+D** (CapsLock+D) to read the current line.
2. Press **Narrator+R** to read output from the current position.
3. For long listings, always redirect to a file and open in Notepad — Notepad is more comfortable for Narrator users than reading directly from the terminal buffer.

Dolphin SuperNova Users

1. After each `cd` command, press **CapsLock+L** to read the current line.
2. Press **CapsLock+Numpad Plus** (say all) to read all output from the current position.
3. For long listings: `dir /B > listing.txt`, then `notepad.exe listing.txt`.

Best Practice for All Screen Readers Always confirm your location after moving:

```
cd FolderName
cd :: Verify your new location
```

Practice Exercises

Exercise 1: Basic Down Navigation

1. `cd %USERPROFILE%`
2. Run `cd` to see location.
3. `dir /B`
4. `cd Documents`
5. Run `cd` to confirm.
6. `dir /B`

Exercise 2: Using Tab Completion

1. `cd %USERPROFILE%`
2. Type `cd Doc` then press **Tab**.
3. Press **Enter**.
4. Run `cd` to confirm.

Exercise 3: Navigating Up

1. Navigate several levels deep.
2. Run `cd` to see full path.
3. Use `cd ..` to go up one level at a time.
4. Run `cd` after each step.

Exercise 4: Absolute Path Navigation

1. Navigate deep (3+ levels).
2. Run `cd %USERPROFILE%` to jump home.
3. Run `cd` to confirm.

Exercise 5: Creating and Navigating a Structure

```
mkdir 3DPractice
cd 3DPractice
mkdir Models
cd Models
mkdir OpenSCAD
cd OpenSCAD
```

Run `cd` at each level. Then navigate back up using only `cd ..`.

Checkpoint Questions

Answer all 10 before moving to CMD-2:

1. What is the difference between `cd Documents` and `cd \`?
2. What does `cd ..` do?
3. How do you go to your home folder from anywhere?

4. What is an absolute path? Give an example.
5. What is a relative path? Give an example.
6. How do you confirm your current location?
7. What does Tab completion do?
8. How would you navigate 3 levels deep, then back home?
9. What is the difference between `.` and `..`?
10. If you are lost, what command should you run first?

Extension Problems

1. Create a folder with spaces in the name (e.g., `My Projects`) and navigate to it using quotes.
2. Use Tab completion to navigate 5+ levels deep without typing full names.
3. Create a script that saves `cd` output at each level to a log file.
4. Navigate to the same destination using both absolute and relative paths from different starting locations.
5. Build a complex folder tree (5+ levels) and navigate using only relative paths.
6. Document the full paths to your 5 most-used folders.
7. Create folders named `01Folder`, `02Folder`, etc. and practice Tab completion through them.
8. Navigate to a folder, save the path with `cd > mypath.txt`, navigate away, then return using the saved path.
9. Challenge: navigate to a destination using only relative paths without running `cd` to check at each step.
10. Create a batch file that navigates to a project folder and lists its contents in one step.

Common Issues

”The system cannot find the path specified”

- Check spelling with `dir /B` to see correct folder names.
- Use Tab completion to avoid typos.
- Use the absolute path: `cd C:\Users\YourName\Documents`.

”I'm lost”

```
cd
```

This always shows your current location.

Tab Completion Not Working

- Type at least one character before pressing Tab.

- Confirm the folder exists with `dir /B`.
- Press Tab again to cycle through multiple matches.

Quick Reference

```
cd :: Show current location
cd FolderName :: Go into a folder
cd .. :: Go up one level
cd ..\.. :: Go up two levels
cd \ :: Go to drive root
cd %USERPROFILE% :: Go to home folder
dir /B :: List folder contents
dir /B /A:D :: List only folders
dir /B /A:F :: List only files
```

Next Steps

Complete all exercises, pass the checkpoint questions, then move to **CMD-2: File & Folder Manipulation**.

CMD-2: File and Folder Manipulation

Duration: 2-2.5 hours (for screen reader users)
Prerequisites: CMD-Pre, CMD-0, CMD-1

Learning Objectives:

- Create, copy, move, and delete files and folders safely
- Use wildcards to operate on multiple files
- Understand dangerous operations and how to stay safe
- Rename files and understand file extensions

Core File Manipulation Commands

Create Folders: `mkdir`

```
mkdir FolderName :: Create a single folder
mkdir Folder1 Folder2 Folder3 :: Create multiple folders at once
mkdir A\B\C :: Create nested folders (A, then B inside A, then C inside B)
```

Create Files: `echo` with Redirection

```
echo This is some content > filename.txt :: Create a file with text content
echo. > emptyfile.txt :: Create an empty file
```

Note: `echo.` (echo followed immediately by a period, no space) creates an empty file. `echo` with a space would put a space character in the file.

Copy Files: `copy`

```
copy source.txt destination.txt :: Copy a file to a new name
copy source.txt backup\ :: Copy a file into a folder
copy *.txt backup\ :: Copy all .txt files to backup folder
```

Copy Folders: `xcopy`

```
xcopy sourcefolder destinationfolder /E /I :: Copy a folder and all its contents
```

- `/E` copies all subdirectories including empty ones.
- `/I` treats the destination as a folder (not a file) if it doesn't exist.

Move and Rename: `move`

```
move oldname.txt newname.txt :: Rename a file
move file.txt folder\ :: Move a file into a folder
move *.txt archive\ :: Move all .txt files to archive folder
```

`move` renames when both source and destination are in the same folder. It moves when the destination is a different folder.

Delete Files: `del`

```
del filename.txt :: Delete one specific file
del *.txt :: Delete all .txt files in current folder
del /Q *.tmp :: Delete quietly (no confirmation prompt)
```

Warning: `del` does not send files to the Recycle Bin. Deleted files are gone immediately.

Delete Folders: `rmdir`

```
rmdir foldername :: Delete an empty folder
rmdir /S /Q foldername :: Delete a folder and ALL its contents (no undo)
```

Warning: `rmdir /S` is permanent. Always verify the folder name before running this command.

Safe File Operations

Rule 1: Always Check Before Deleting

```
dir /B :: See exactly what is here
dir /B *.txt :: See exactly which .txt files exist
del *.txt :: Only then delete
```

Rule 2: Make Backups First

```
mkdir backup
copy *.txt backup\ :: Copy all .txt files to a backup folder first
del *.txt :: Now safe to delete originals
```

Rule 3: Test on One File First

```
del test-one-file.txt :: Test deletion on a single file
dir /B :: Confirm it is gone
```

Only then proceed with bulk operations.

Using Wildcards

Wildcards let you operate on multiple files matching a pattern.

* Wildcard (Match Any Number of Characters)

```
dir /B *.txt :: List all files ending in .txt
copy *.scad backup\ :: Copy all .scad files to backup
del *.tmp :: Delete all .tmp files
```

? Wildcard (Match Exactly One Character)

```
dir /B file?.txt :: Matches file1.txt, file2.txt, filea.txt, etc.
copy model?.scad models\ :: Copy model1.scad, model2.scad, etc.
```

Practical Examples

Example 1: Create a Project Structure

```
mkdir 3DProjects
cd 3DProjects
mkdir Models Prints Documentation Backups
dir /B /A:D
```

Example 2: Backup Before Editing

```
copy project.scad project-backup.scad :: Create a backup copy
:: (edit project.scad)
copy project.scad project-v2.scad :: Save a new versioned copy
```

Example 3: Organize Files by Type

```
mkdir scad-files
mkdir text-files
move *.scad scad-files\
move *.txt text-files\
dir /B /A:D
```

Practice Exercises

Exercise 1: Create a Folder Structure

1. `mkdir practice-session`
2. `cd practice-session`
3. `mkdir files documents models`
4. `dir /B /A:D`

Goal: Create organized folder structures confidently.

Exercise 2: Create and Copy Files

1. `echo Hello World > test.txt`
2. `type test.txt`
3. `copy test.txt test-backup.txt`
4. `dir /B *.txt`
5. `type test-backup.txt`

Goal: Create and copy files without errors.

Exercise 3: Safe Deletion Practice

1. `echo content > file1.txt`
2. `echo content > file2.txt`
3. `echo content > file3.txt`
4. `dir /B *.txt` (confirm all three exist)
5. `del file1.txt` (delete just one)
6. `dir /B *.txt` (confirm only two remain)

Goal: Practice the ”check, then delete” pattern.

Exercise 4: Wildcard Operations

1. Create three files: `echo a > doc1.txt`, `echo b > doc2.txt`, `echo c > doc3.txt`
2. `dir /B *.txt`
3. `mkdir archive`
4. `copy *.txt archive\`
5. `cd archive`
6. `dir /B *.txt`

Goal: Comfortable using wildcards for bulk operations.

Screen Reader Tips

NVDA Users After file operations, run `dir /B` and press **NVDA+Down Arrow** to verify results. Redirect output to a file for complex verification: `dir /B > check.txt`, then `notepad.exe check.txt`.

JAWS Users After file operations, run `dir /B` and press **Insert+Down Arrow** to read results. For complex output: `dir /B > check.txt`, then `notepad.exe check.txt`.

Windows Narrator Users After file operations, run `dir /B` and press **Narrator+R** to read from the current position. Redirect to a file and open in Notepad for long output.

Dolphin SuperNova Users After file operations, run `dir /B` and press **CapsLock+Numpad Plus** to read results. For long output: `dir /B > check.txt`, then `notepad.exe check.txt`.

General Tip for All Screen Readers After any `copy`, `move`, or `del` operation, always verify with `dir /B` to confirm the expected result. Never assume an operation worked without checking.

Checkpoint Questions

1. How do you create a folder?
2. How do you copy a file?
3. How do you rename a file?
4. How do you delete a file safely (checking first)?
5. What does `*` match in a wildcard pattern?
6. What does `?` match in a wildcard pattern?
7. How would you copy all .scad files to a backup folder?
8. What should you always do before deleting?
9. How do you delete a folder and all its contents?
10. Why is it important to create a backup before editing?

Extension Problems

1. Create a nested folder structure (5+ levels) using a single `mkdir` command with backslashes.
2. Create 10 files and organize them into subfolders using `move` with wildcards.
3. Create a date-based backup strategy: copy all files to a folder named with today's date.
4. Use wildcards to select specific file types and copy them to separate organized folders.
5. Create a file-renaming system (file001.txt, file002.txt, etc.) and practice moving them between folders.

Next Steps

Complete all exercises, pass the checkpoint questions, then move to **CMD-3: Input, Output & Redirection**.

CMD-3: Input, Output, and Piping

Duration: 1 class period **Prerequisite:** CMD-2 (File and Folder Manipulation)

Learning Objectives

By the end of this lesson, you will be able to:

- Use `echo` to print text to the screen
- Use `type` to read file contents
- Use > to redirect output into a file
- Use | (pipe) to send one command's output to another
- Copy output to the clipboard with `clip`
- Open files with a text editor from the command line

Commands Covered

Command	What It Does
`echo text`	Print text to the screen
`type filename`	Print the contents of a file
`> filename`	Redirect output into a file (overwrites)
`>> filename`	Append output to a file (adds to end)
|	Pipe - send output from one command to the next
`clip`	Copy piped input to the Windows clipboard
`notepad filename`	Open a file in Notepad

`echo` - Printing Text

`echo` prints text to the screen. It is useful for testing, for writing text into files, and for understanding how piping works.

```
echo Hello, World
echo This is a test
```

`type` - Reading Files

`type` prints the contents of a file to the screen.

```
:: Read a text file
type %USERPROFILE%\Documents\notes.txt

:: Read an OpenSCAD file
type %USERPROFILE%\Documents\OpenSCAD_Projects\project0.scad
```

With a long file, use `type filename | more` to read it page by page (press `Space` to advance, `Q` to quit).

> - Redirecting Output to a File

The > symbol redirects output from the screen into a file instead.

```
:: Create a file with a single line
echo Author: Your Name > header.txt

:: Confirm the file was created and has content
type header.txt
```

Warning: > overwrites the file if it already exists. Use >> to append instead:

```
echo Date: 2025 >> header.txt
echo Project: Floor Marker >> header.txt
type header.txt
```

| - Piping

The pipe symbol | sends the output of one command to the input of the next. This lets you chain commands together.

```
:: List files and send the list to clip (copies to clipboard)
dir /B | clip

:: Now paste with Ctrl + V anywhere
```

```
:: Search within a file's contents using find
type project0.scad | find "cube"
```

`clip` - Copying to Clipboard

`clip` takes whatever is piped to it and puts it on the Windows clipboard.

```
:: Copy your current directory path to the clipboard
cd | clip

:: Copy a file listing to clipboard
dir /B | clip

:: Copy the contents of a file to clipboard
type notes.txt | clip
```

After any of these, press `Ctrl + V` in any application to paste.

Opening Files in Notepad

```
:: Open a file in Notepad
notepad %USERPROFILE%\Documents\notes.txt

:: Open a .scad file
notepad %USERPROFILE%\Documents\OpenSCAD_Projects\project0.scad

:: Create a new file and open it
echo. > new_notes.txt
notepad new_notes.txt
```

Step-by-step Tasks

1. Create `practice.txt` with three lines using `echo` and >/>>.
2. Read the file with `type practice.txt`.
3. Pipe the file into `find` to search for a word.
4. Copy the file contents to clipboard with `type practice.txt | clip`.
5. Redirect `dir /B` into `list.txt` and open it in Notepad.

Checkpoints

- After step 3 you should be able to find a keyword using piping.

Quiz - Lesson CMD.3

1. What is the difference between > and >>?
2. What does the pipe | do?
3. How do you copy output to the clipboard?
4. How would you page through long output?
5. How do you suppress output (send it nowhere)?
6. True or False: The pipe operator | connects the output of one command to the input of another.
7. Explain why redirecting output to a file is useful for screen reader users.
8. Write a command that would search for the word "sphere" in all `.scad` files in a directory.
9. How would you count the number of lines in a file using CMD piping?
10. Describe a practical scenario in 3D printing where you would pipe or redirect command output.
11. What would be the difference in output between `echo test > file.txt` (run twice) vs `echo test >> file.txt` (run twice)? Show the expected file contents.
12. Design a three-step piping chain: read a file, filter for specific content, and save the results; explain what each pipe does.

13. You have a 500-line `.scad` file and need to find all instances of `sphere()` and count them. Write the command.
14. Explain how `clip` is particularly valuable for screen reader users when working with file paths or long output strings.
15. Describe how you would use pipes and redirection to create a timestamped backup report of all `.stl` files in a 3D printing project folder.

Extension Problems

1. Use piping to count lines in a file (hint: `type file.txt | find /C /V ""`).
2. Save a long `dir /B` output and search it with `find`.
3. Chain multiple pipes to filter and then save results.
4. Practice copying different command outputs to clipboard and pasting.
5. Create a small batch script that generates a report (counts of files by extension).
6. Build a data processing pipeline: read a text file, filter rows, and export results; document each step.
7. Write a batch script that pipes directory listing to count occurrences of each file extension; create a summary report.
8. Create a log analysis command: read a log file, filter for errors, and save matching lines to a separate error log.
9. Design a piping workflow for 3D printing file management: find `.stl` files, extract their names, and generate a report.
10. Develop a reusable piping pattern library: create batch scripts for common filtering, sorting, and reporting patterns; test each with different inputs.
11. Build a complex filter pipeline: read a `.scad` file, extract lines containing specific geometry commands, count each type, and output a summary to both screen and file.
12. Create an interactive filtering tool: build a batch script that accepts a search term, pipes through multiple filters, and displays paginated results.
13. Develop a performance analysis tool: use piping to combine file listing, metadata extraction, and statistical reporting; export results to a dated report file.
14. Implement a comprehensive error-handling pipeline: read output, catch errors, log them separately, and generate a summary of successes vs failures.
15. Design and execute a real-world project backup workflow: use piping to verify file existence, count files by type, generate a backup manifest, and create audit logs — all in one integrated command pipeline.

References

- Microsoft. (2024). *Using redirection operators in CMD*. https://example.com
- Microsoft. (2024). *FIND command reference*. https://example.com

- Microsoft. (2024). *CMD pipeline concepts*. https://example.com

Helpful Resources

- Using Redirection in CMD
- Piping and FIND
- TYPE Command Reference
- FINDSTR for Advanced Searching
- CMD Pipeline Concepts

CMD-4: Environment Variables, PATH, and Aliases

Estimated time: 30-45 minutes

Learning Objectives

- Read environment variables with `%VARNAME%`
- Inspect and verify programs in the `PATH`
- Create temporary aliases with `doskey` and understand making them persistent via a startup script

Materials

- Command Prompt

Step-by-step Tasks

1. Show your username and home path with `echo %USERNAME%` and `echo %USERPROFILE%`.
2. Inspect `echo %PATH%` and identify whether `openscad` or `code` would be found.
3. Run `where openscad` and note the result.
4. Create a temporary alias: `doskey preview=openscad $*` and test `preview myfile.scad`.
5. Open your startup script (`notepad.exe autorun.bat`) and add the doskey line to make it persistent (advanced).

Checkpoints

- After step 3 you can determine whether a program will be found by `PATH`.

Quiz - Lesson CMD.4

1. How do you print an environment variable?
2. What is the purpose of `PATH`?
3. How do you check whether `openscad` is available?

4. How do you create a temporary alias in CMD?
5. Where would you make an alias permanent?
6. True or False: Environment variable names in CMD are case-sensitive.
7. Explain why having a program in your PATH is useful compared to always using its full file path.
8. Write a command that would create a doskey alias called `slicer` for the OpenSCAD executable.
9. What file or technique would you use to make a doskey alias persist across CMD sessions?
10. Describe a practical benefit of using the `%TEMP%` directory for temporary files in a 3D printing workflow.
11. You have a custom batch script at `C:\Scripts\backup_models.bat` that you want to run from anywhere as `backup-now`. What steps would you take to make this work?
12. Explain the difference between setting an environment variable in the current session with `set` vs. using `setx` for permanence.
13. Design a strategy for managing multiple 3D printing projects, each with different tool paths and directories; show how to structure environment variables for each.
14. If a program is not found by `where`, what are the possible reasons, and how would you troubleshoot?
15. Describe how you would verify that your CMD autorun script is loading correctly and how to debug issues if aliases or environment variables don't appear after opening a new CMD session.

Extension Problems

1. Add a folder to PATH for a test program (describe steps; do not change system PATH without admin).
2. Create a short autorun snippet that sets two aliases and test re-opening CMD.
3. Use `where` to list the path for several common programs.
4. Explore `%TEMP%` and create a file there.
5. Save a copy of your current PATH to a text file and examine it in your editor.
6. Create a CMD autorun script that loads custom aliases and environment variables for your 3D printing workflow; test it in a new session.
7. Build a ”project profile” batch script that sets environment variables for CAD, slicing, and print directories; switch between profiles for different projects.
8. Write a batch script that audits your current environment variables and creates a summary report of what's set and why.
9. Design a custom alias system using doskey for common 3D printing commands; document the aliases and their purposes.
10. Create a profile migration guide: document how to export and import your CMD aliases and variables across machines for consistent workflows.
11. Implement a safe PATH modification script: create a utility that allows you

to add/remove directories from PATH for the current session only; show how to make it permanent with `setx`.
12. Build a comprehensive autorun framework: create separate `.bat` files for aliases, environment variables, and helper macros; have your main autorun load all of them.
13. Develop an environment validation tool: write a batch script that checks whether all required programs (OpenSCAD, slicers, etc.) are accessible via PATH; report findings and suggest fixes.
14. Create a project-switching alias system: design a batch script that changes all environment variables and aliases based on the current project; test switching between multiple projects.
15. Build a troubleshooting guide: create a batch script that exports your current environment state (variables, aliases, PATH) to a timestamped file, allowing you to compare states before and after changes.

References

- Microsoft. (2024). *Environment variables in CMD*. https://example.com
- Microsoft. (2024). *DOSKEY command reference*. https://example.com
- Microsoft. (2024). *WHERE command reference*. https://example.com

Helpful Resources

- Environment Variables in CMD
- Understanding the PATH Variable
- DOSKEY Alias Reference
- WHERE Command for Locating Programs
- SETX for Permanent Variables

CMD-5: Filling in the Gaps - Control Flow, Startup Scripts, and Useful Tricks

Estimated time: 30-45 minutes

Learning Objectives

- Use history and abort commands (`doskey /history`, `F7`, `Ctrl+C`)
- Inspect and edit your CMD autorun script for persistent settings
- Run programs by full path using the `start` or call operator
- Handle common screen reader edge cases when using the terminal

Materials

- Command Prompt and an editor (Notepad/VS Code)

Step-by-step Tasks

1. Run several simple commands (e.g., `cd`, `dir /B`, `echo hi`) then press `F7` or run `doskey /history` to view them.
2. Use `F8` to search back through history, or use the up-arrow to re-run a previous command.
3. Practice aborting a long-running command with `Ctrl + C` (for example, `ping 8.8.8.8`).
4. Open your autorun script: `notepad.exe autorun.bat`; if it doesn't exist, create it with `echo. > autorun.bat`.
5. Add a persistent alias line to your autorun script (example: `doskey preview=openscad $*`), save, and configure CMD to use it by registering it in the registry (advanced).

Checkpoints

- After step 2 you can re-run a recent command from history.
- After step 5 your alias should persist across CMD sessions.

Quiz - Lesson CMD.5

1. How do you view the command history in CMD?
2. Which key combination aborts a running command?
3. What does `echo %CMDCMDLINE%` show?
4. How does the `start` command help run executables?
5. What is one strategy if terminal output stops being announced by your screen reader?
6. True or False: Using `Ctrl+C` permanently deletes any files created by the command you abort.
7. Explain the difference between pressing `F7` and running `doskey /history` in CMD.
8. If you add a doskey macro to your autorun script but it doesn't take effect after opening a new CMD window, what should you verify?
9. Write a command that would run a program at the path `C:\Program Files\OpenSCAD\openscad.exe` directly.
10. Describe a practical workflow scenario where having keyboard shortcuts (doskey macros) in your autorun script would save time.
11. Explain how to re-run the 5th command from your history using `F7` and selection, and what would happen if that command had file operations (creates/deletes).
12. Design an autorun initialization strategy that separates utilities for different projects; explain how you would switch between them.
13. Walk through a troubleshooting workflow: your screen reader stops announcing output after running a long command. What steps would you take to diagnose and resolve the issue?
14. Create a safety checkpoint system: before any destructive operation (mass

delete, overwrite), how would you use autorun macros and history to verify the command is correct?

15. Develop a comprehensive capstone scenario: integrate everything from CMD-0 through CMD-5 (navigation, file operations, piping, environment setup, history) to design an automated 3D printing project workflow with error handling and logging.

Extension Problems

1. Add a doskey macro and an environment variable change to your autorun script and document the behavior after reopening CMD.
2. Create a short batch script that automates creating a project folder and an initial `.scad` file.
3. Experiment with running OpenSCAD by full path using `start` and by placing it in PATH; compare results.
4. Practice redirecting `help` output to a file and reading it in Notepad for screen reader clarity.
5. Document three screen reader troubleshooting steps you used and when they helped.
6. Build a comprehensive CMD autorun script that includes aliases, environment variables, and helper macros for your 3D printing workflow.
7. Create a batch script that troubleshoots common CMD issues (missing commands, permission errors, command not found); test at least three scenarios.
8. Write a batch script that coordinates multiple tasks: creates a project folder, starts OpenSCAD, and opens a notes file.
9. Design a screen-reader accessibility guide for CMD: document commands, outputs, and accessible navigation patterns.
10. Develop an advanced CMD workflow: implement error handling, logging, and confirmation prompts for risky operations.
11. Implement an ”undo” system using history: create a batch script that logs destructive commands (`del`, `move`, `copy /Y`) and allows you to review the last operation.
12. Build an autorun debugger: create a script that compares two CMD sessions' environment states (variables, aliases, macros) to identify what loaded/failed to load.
13. Develop a multi-project autorun manager: design a system where you can switch entire environments (paths, aliases, variables) for different 3D printing projects by running a single script.
14. Create a comprehensive accessibility analyzer: write a batch script that tests whether key CMD commands produce screen-reader-friendly output; document workarounds for commands that don't.
15. Design a complete capstone project: build an integrated automation suite that manages a 3D printing workflow (project setup, file organization, CAD/slicing tool automation, output logging, error recovery, and audit trails) with full error handling and documentation.

References

- Microsoft. (2024). *CMD history and recall functionality*. https://example.com
- Microsoft. (2024). *Using CMD autorun scripts*. https://example.com
- Microsoft. (2024). *The START command for running executables*. https://example.com

Helpful Resources

- DOSKEY History and Recall
- CMD Autorun Scripts
- START Command Reference
- CMD History Navigation
- Screen Reader Tips for CMD

CMD-6: Advanced Terminal Techniques - Batch Scripts, Functions & Professional Workflows

Duration: 4-4.5 hours (for screen reader users) **Prerequisites:** Complete CMD-0 through CMD-5 **Skill Level:** Advanced intermediate

This lesson extends CMD skills to professional-level workflows. You'll learn to automate complex tasks, write reusable batch scripts, and integrate tools for 3D printing workflows.

Learning Objectives

By the end of this lesson, you will be able to:

- Create and run batch files (.bat files)
- Write subroutines that accept parameters
- Use loops to repeat tasks automatically
- Automate batch processing of 3D models
- Debug batch scripts when something goes wrong
- Create professional workflows combining multiple tools

Batch Script Basics

What's a Batch Script? A batch file (`.bat`) contains multiple CMD commands that run in sequence. Instead of typing commands one by one, you put them in a file and run them all at once.

Why use batch scripts?

- Repeatability: Run the same task 100 times identically
- Documentation: Commands are written down for reference
- Complexity: Combine many commands logically
- Automation: Schedule scripts to run automatically

Creating Your First Batch Script

Step 1: Open a text editor

```
notepad.exe my-first-script.bat
```

Step 2: Type this script

```
@echo off
:: This is a comment - screen readers will read it
echo Script is running!
cd
dir /B
echo Script is done!
```

Step 3: Save the file

- In Notepad: Ctrl+S
- Make sure filename ends in `.bat`
- Save in an easy-to-find location (like Documents)

Step 4: Run the script

```
my-first-script.bat
```

What happens: CMD runs each command in sequence and shows output.

Important: `@echo off` On some scripts, each command is echoed before running. `@echo off` at the top suppresses this, showing only output — not the commands themselves.

```
@echo off
:: Now only output is shown, not each command
echo Hello!
```

Variables and Parameters

Using Variables Variables store values you want to use later.

Example script:

```
@echo off
set mypath=C:\Users\YourName\Documents
cd %mypath%
echo I am now in:
cd
dir /B
```

Breaking it down:

- `set mypath=...` assigns the variable
- `%mypath%` uses the variable (wraps it in `%`)
- Variables in CMD are always referenced with `%VARNAME%`

Subroutines with Parameters A subroutine is reusable code that you can call with different inputs using `:label` and `call`.

Example: A subroutine that lists files in a folder

```
@echo off
call :ListFolder "C:\Users\YourName\Documents"
call :ListFolder "C:\Users\YourName\Downloads"
goto :eof

:ListFolder
echo Contents of: %~1
cd /D %~1
dir /B
goto :eof
```

What's happening:

- `:ListFolder` marks the start of the subroutine
- `%~1` is the first argument passed to the subroutine
- `call :ListFolder "path"` calls the subroutine with a path
- `goto :eof` exits the subroutine (or script)

Screen reader tip: When you call a subroutine, CMD will announce the results just like any command.

Loops - Repeating Tasks

Loop Over Files Imagine you have 10 SCAD files and want to print their names. You could do it 10 times manually, or use a loop.

Example: Print every .scad file in a folder

```
@echo off
for %%f in (*.scad) do (
    echo === File: %%f ===
    type %%f
    echo.
)
```

What's happening:

- `for %%f in (*.scad) do` iterates over each `.scad` file
- `%%f` is the loop variable (use `%f` in interactive CMD, `%%f` in batch files)
- Inside the parentheses, do something with each file

Loop with a Counter **Example: Do something 5 times**

```
@echo off
for /L %%i in (1,1,5) do (
    echo This is iteration number %%i
)
```

What's happening:

- `for /L %%i in (start,step,end)` counts from start to end
- `%%i` is the counter variable

Real-World Example - Batch Processing SCAD Files

Scenario You have 10 OpenSCAD (.scad) files in a folder. You want to:

1. List them all
2. Check how many there are
3. For each one, verify it exists

The Script

```
@echo off
set scadFolder=C:\Users\YourName\Documents\3D_Projects
set count=0

echo Processing SCAD files in: %scadFolder%
echo.

for %%f in (%scadFolder%\*.scad) do (
    if exist "%%f" (
        echo Found: %%~nxf
        set /A count+=1
    ) else (
        echo Missing: %%~nxf
    )
)

echo.
echo Total files found: %count%
echo Batch processing complete!
```

Breaking it down:

- `set scadFolder=...` = where to look
- `for %%f in (...\*.scad)` = find all .scad files
- `if exist "%%f"` = check if file exists
- `%%~nxf` = just the filename with extension (not the full path)
- `set /A count+=1` = increment the counter

Running the Script

1. Save as `batch-process.bat`
2. Edit `scadFolder` to match your real folder
3. Run it:

```
batch-process.bat
```

Screen reader output:

```
Processing SCAD files in: C:\Users\YourName\Documents\3D_Projects

 Found: model1.scad
 Found: model2.scad
 Found: model3.scad
[... more files ...]

Total files found: 10
Batch processing complete!
```

Error Handling

Checking Error Levels What if something goes wrong? Use `errorlevel` checks:

Example:

```
@echo off
type C:\nonexistent\path\file.txt
if %errorlevel% neq 0 (
    echo Error: Could not read file
    echo Errorlevel: %errorlevel%
)
```

What's happening:

- After any command, `%errorlevel%` holds `0` for success or non-zero for failure
- `if %errorlevel% neq 0` checks for failure
- Handle the error gracefully

Screen reader advantage: Errors are announced clearly instead of crashing silently.

Validating Input **Example: Make sure a folder exists before processing**

```
@echo off
set folderPath=C:\Users\YourName\Documents

if not exist "%folderPath%\" (
    echo Error: Folder does not exist: %folderPath%
    goto :eof
)

echo Processing folder: %folderPath%
dir /B "%folderPath%"
```

What's happening:

- `if not exist "%folderPath%\"` checks if folder exists (trailing `\` ensures it's a folder)

- `goto :eof` exits the script early if error

Debugging Batch Scripts

Common Errors and Solutions

Error 1: "Command not found" **Cause:** Typo in command name or program not in PATH

Fix: Check spelling and verify PATH

```
:: Wrong:
dri /B

:: Correct:
dir /B
```

Error 2: "Variable is empty" **Cause:** Variable was never set or has a typo

Fix: Make sure variable is set before using it

```
set myvar=hello
echo %myvar%
```

Error 3: "The system cannot find the path specified" **Cause:** Wrong folder path

Fix: Verify path exists

```
:: Check if path exists:
if exist "C:\Users\YourName\Documents\" echo Path exists
```

Error 4: "Access denied" **Cause:** Don't have permission

Fix: Run CMD as administrator

- Right-click Command Prompt -> Run as administrator

Debugging Technique: Trace Output Add `echo` statements to track what's happening:

```
@echo off
set path_var=C:\Users\YourName\Documents
echo Starting script. Path is: %path_var%

for %%f in (%path_var%\*) do (
    echo Processing: %%f
    :: Do something with %%f
    echo Done with: %%f
)

echo Script complete
```

Your screen reader will announce each step, so you know where errors happen.

Creating Professional Workflows

Example 1: Automated Project Setup **Scenario:** You start a new 3D printing project regularly. Instead of creating folders manually:

```
@echo off
set /p projectName=Enter project name:
set baseFolder=C:\Users\YourName\Documents\3D_Projects
set projectFolder=%baseFolder%\%projectName%

:: Create folder structure
mkdir "%projectFolder%"
mkdir "%projectFolder%\designs"
mkdir "%projectFolder%\output"
mkdir "%projectFolder%\notes"

:: Create a README
echo # %projectName% > "%projectFolder%\README.txt"
echo. >> "%projectFolder%\README.txt"
echo Created: %date% >> "%projectFolder%\README.txt"
echo. >> "%projectFolder%\README.txt"
echo ## Designs >> "%projectFolder%\README.txt"
echo All .scad files go here. >> "%projectFolder%\README.txt"
echo. >> "%projectFolder%\README.txt"
echo ## Output >> "%projectFolder%\README.txt"
echo STL and other exports go here. >> "%projectFolder%\README.txt"

echo Project setup complete: %projectFolder%
```

What it does:

- Prompts for a project name
- Creates folder structure for a new project
- Sets up subfolders for designs, output, notes
- Creates a README file automatically

Example 2: Batch File Verification **Scenario:** Before processing, verify all required files exist:

```
@echo off
set projectFolder=C:\Users\YourName\Documents\3D_Projects\MyKeychain
set allGood=1

for %%i in (README.txt designs output notes) do (
    if exist "%projectFolder%\%%i" (
        echo Found: %%i
    ) else (
        echo Missing: %%i
        set allGood=0
    )
)

if %allGood%==1 (
    echo All checks passed!
) else (
    echo Some files are missing!
)
```

Screen Reader Tips for Batch Scripts

Making Script Output Readable **Problem:** Script runs but output scrolls too fast or is hard to follow

Solution 1: Save to file

```
my-script.bat > output.txt
notepad.exe output.txt
```

Solution 2: Use echo with clear sections

```
echo ========== STARTING ==========
echo.
:: ... script ...
echo.
echo ========== COMPLETE ==========
```

Solution 3: Pause between major sections

```
echo Pausing... Press any key to continue
pause > nul
```

Your screen reader will announce the pause, give you time to read output.

Announcing Progress **For long-running scripts:**

```
@echo off
set count=0
for %%f in (*.scad) do (
    set /A count+=1
    echo Processing file %%f
)
echo All %count% files processed!
```

Practice Exercises

Exercise 1: Your First Batch Script **Goal:** Create and run a simple batch script

Steps:

1. Create file: `notepad.exe hello.bat`
2. Type:

   ```
   @echo off
   echo Hello from my first CMD batch script!
   cd
   dir /B
   ```

3. Save and run: `hello.bat`

Checkpoint: You should see output for each command.

Exercise 2: Script with a Variable **Goal:** Use a variable to make the script flexible

Steps:

1. Create file: `notepad.exe smart-listing.bat`
2. Type:

```
@echo off
set targetFolder=C:\Users\YourName\Documents
echo Listing contents of: %targetFolder%
dir /B "%targetFolder%"
```

3. Edit `targetFolder` to a real folder on your computer
4. Run: `smart-listing.bat`

Checkpoint: You should see listing of that specific folder.

Exercise 3: Subroutine **Goal:** Create a reusable subroutine

Steps:

1. Create file: `notepad.exe navigate.bat`
2. Type:

```
@echo off
call :GoTo "C:\Users\YourName\Documents"
call :GoTo "C:\Users\YourName\Downloads"
goto :eof

:GoTo
if exist "%~1\" (
    cd /D "%~1"
    echo Now in:
    cd
    echo Contents:
    dir /B
) else (
    echo Path does not exist: %~1
)
goto :eof
```

3. Run: `navigate.bat`

Checkpoint: Both subroutine calls should work, showing contents of each folder.

Exercise 4: Loop **Goal:** Use a loop to repeat an action

Steps:

1. Create file: `notepad.exe repeat.bat`
2. Type:

```
@echo off
echo Demonstrating a loop:
for /L %%i in (1,1,5) do (
    echo Iteration %%i: Hello!
)
echo Loop complete!
```

3. Run: `repeat.bat`

Checkpoint: Should print "Iteration 1" through "Iteration 5".

Exercise 5: Real-World Script **Goal:** Create a useful script for a real task

Steps:

1. Create a folder: `mkdir C:\Users\YourName\Documents\TestFiles`
2. Create some test files:

```
echo test > C:\Users\YourName\Documents\TestFiles\file1.txt
echo test > C:\Users\YourName\Documents\TestFiles\file2.txt
echo test > C:\Users\YourName\Documents\TestFiles\file3.txt
```

3. Create script: `notepad.exe report.bat`
4. Type:

```
@echo off
set folder=C:\Users\YourName\Documents\TestFiles
set count=0

echo === FILE REPORT ===
echo Folder: %folder%
echo.
echo Files:
for %%f in ("%folder%\*") do (
    echo - %%~nxf
    set /A count+=1
)
echo.
echo Total: %count% files
echo === END REPORT ===
```

5. Run: `report.bat`

Checkpoint: Should show report of all files in the test folder.

Quiz - Lesson CMD.6

1. What is a CMD batch script?
2. What file extension do CMD batch scripts use?
3. What is a variable in a batch script and how do you create one?
4. What is a subroutine (`:label`) and why would you use one?
5. How do you run a batch script?
6. What is a `for` loop and what does `for %%f in (*.scad) do` do?

7. What does `if exist` do?
8. How do you handle errors in a batch script?
9. When would you use `if %errorlevel% neq 0`?
10. What technique makes batch script output readable for screen readers?

Extension Problems

1. **Auto-Backup Script:** Create a batch script that copies all files from one folder to another, announcing progress
2. **File Counter:** Write a subroutine that counts files by extension (.txt, .scad, .stl, etc.)
3. **Folder Cleaner:** Batch script that deletes files older than 30 days (with user confirmation)
4. **Project Template:** Subroutine that creates a complete project folder structure with all needed files
5. **Batch Rename:** Script that renames all files in a folder according to a pattern
6. **Log Generator:** Create a script that records what it does to a log file for later review
7. **Scheduled Task:** Set up a batch script to run automatically every day at a specific time
8. **File Verifier:** Check that all SCAD files in a folder have corresponding STL exports
9. **Report Generator:** Create a summary report of all projects in a folder
10. **Error Tracker:** Script that lists all commands that had errors and logs them with timestamps

---

Important Notes

- **Always test scripts on small sets of files first** before running them on important data
- **Save your work regularly** — use version naming if possible
- **Test error handling** — make sure errors don't crash silently
- **Document your scripts** — use `::` comments so you remember what each part does
- **Backup before batch operations** — if something goes wrong, you have the original

---

References

- **Microsoft CMD Batch Scripting Guide:** https://example.com

- **FOR Loop Documentation:** https://example.com
- **IF Statement Reference:** https://example.com
- **SET Variable Reference:** https://example.com

Next Steps: After mastering this lesson, explore advanced batch scripting, scheduled tasks, and 3D printing integration in the main curriculum.

CMD Unit Test - Comprehensive Assessment

Estimated time: 60-90 minutes

Key Learning Outcomes Assessed

By completing this unit test, you will demonstrate:

1. Understanding of file system navigation and path concepts
2. Proficiency with file and folder manipulation commands
3. Ability to redirect and pipe command output
4. Knowledge of environment variables and aliases
5. Screen-reader accessibility best practices in terminal environments
6. Problem-solving and command chaining skills

Target Audience

Users who have completed CMD-0 through CMD-6 and need to demonstrate mastery of Command Prompt fundamentals.

Instructions

Complete all sections below. For multiple choice, select the best answer. For short answers, write one to two sentences. For hands-on tasks, capture evidence (screenshots or output files) and submit alongside your answers.

Part A: Multiple Choice Questions (20 questions)

Select the best answer for each question. Each question is worth 1 point.

1. What is the primary purpose of the `PATH` environment variable?
    - A) Store your home directory location
    - B) Tell the shell where to find executable programs
    - C) Configure the visual appearance of the terminal
    - D) Store the current working directory name
2. Which command shows your current working directory in CMD?

- A) `dir /B`
- B) `cd` (with no arguments)
- C) `pwd`
- D) `whoami`

3. What does `%USERPROFILE%` represent in CMD?
    - A) The root directory
    - B) The current directory
    - C) The parent directory
    - D) Your home directory
4. How do you list only file names (not full details) in a screen-reader-friendly way?
    - A) `dir`
    - B) `dir /B`
    - C) `dir /L`
    - D) `type /B`
5. Which command creates a new empty file in CMD?
    - A) `mkdir filename`
    - B) `echo. > filename`
    - C) `touch filename`
    - D) `new filename`
6. What is the difference between > and >>?
    - A) > redirects to file, >> displays on screen
    - B) > overwrites a file, >> appends to a file
    - C) They do the same thing
    - D) > is for text, >> is for binary
7. What does the pipe operator `|` do?
    - A) Creates a folder
    - B) Sends the output of one command to the input of another
    - C) Deletes files matching a pattern
    - D) Lists all processes
8. Which command copies a file in CMD?
    - A) `move`
    - B) `del`
    - C) `copy`
    - D) `cd`
9. How do you rename a file from `oldname.txt` to `newname.txt` in CMD?
    - A) `copy oldname.txt newname.txt`
    - B) `move oldname.txt newname.txt`
    - C) `rename oldname.txt newname.txt`

- D) Both B and C are correct

10. What is the purpose of `find` in CMD piping?
    - A) Find files in a directory
    - B) Search for text patterns within output
    - C) Find a string to copy to clipboard
    - D) Find which shell to use
11. Which key allows you to autocomplete a path in CMD?
    - A) `Ctrl + A`
    - B) `Ctrl + E`
    - C) `Tab`
    - D) `Space`
12. How do you copy text to the Windows clipboard from CMD?
    - A) `type filename > clipboard`
    - B) `type filename | clip`
    - C) `copy filename`
    - D) `type filename | paste`
13. What does `where openscad` do?
    - A) Opens the OpenSCAD application
    - B) Gets help about the openscad command
    - C) Locates the full path of the openscad executable
    - D) Lists all available commands
14. Which wildcard matches any single character?
    - A) `*`
    - B) `?`
    - C) `%`
    - D) `#`
15. What is the purpose of the `start` command?
    - A) Run a script or executable, optionally in a new window
    - B) Execute all commands in parallel
    - C) Combine multiple commands
    - D) Create an alias
16. How do you create a temporary alias (macro) in CMD?
    - A) `alias preview='openscad'`
    - B) `doskey preview=openscad $*`
    - C) `set preview=openscad`
    - D) `macro preview openscad`
17. How can doskey macros be made to persist across CMD sessions?
    - A) They are automatically saved

- B) By adding them to a startup batch script registered as an autorun
- C) By using the Windows Control Panel
- D) Doskey macros cannot be made permanent

18. How do you abort a long-running command in CMD?
    - A) Press `Escape`
    - B) Press `Ctrl + X`
    - C) Press `Ctrl + C`
    - D) Press `Alt + F4`
19. What command shows the history of previously run commands in CMD?
    - A) `history`
    - B) `doskey /history`
    - C) `F7` (opens history popup)
    - D) Both B and C are correct
20. How do you make an environment variable permanent in CMD (for all future sessions)?
    - A) Use `set` in the terminal every time
    - B) Use `setx` to write it to the registry
    - C) Use the Windows Control Panel only
    - D) Environment variables cannot be made permanent in CMD

Part B: Short Answer Questions (10 questions)

Answer each question in one to two sentences. Each question is worth 2 points.

1. Explain the difference between absolute and relative paths. Give one example of each.
2. Why is `dir /B` preferred over `dir` for screen reader users? Describe what flag you would add to list only files.
3. What is the purpose of redirecting output to a file, and give an example of when you would use > instead of >>?
4. Describe what would happen if you ran `rmdir /S /Q C:\Documents\my_folder` and why this command should be used carefully.
5. How would you search for all files with a `.scad` extension in your current directory? Write the command.
6. Explain what happens when you pipe the output of `dir /B` into `clip`. What would you do next?
7. What is an environment variable, and give one example of how you might use it in CMD.

8. If a program is not in your `PATH`, what two methods could you use to run it from CMD?
9. Describe how you would open a file in Notepad and also add a line to it from CMD.
10. What is one strategy you would use if your screen reader stops announcing terminal output while using CMD?

Part C: Hands-On Tasks (10 tasks)

Complete each task and capture evidence (screenshots, output files, or command transcripts). Each task is worth 3 points.

Tasks 1-5: File System and Navigation

1. Create a folder structure `%USERPROFILE%\Documents\CMD_Assessment\Projects` using a single command. Capture the `dir /B` output showing the creation.
2. Create five files named `project_1.scad`, `project_2.scad`, `project_3.txt`, `notes_1.txt`, and `notes_2.txt` inside the `Projects` folder. Use wildcards to list only `.scad` files, then capture the output.
3. Copy the entire `Projects` folder to `Projects_Backup` using `xcopy /E /I`. Capture the `dir /B` output showing both folders exist.
4. Move (rename) `project_1.scad` to `project_1_final.scad`. Capture the `dir /B` output showing the renamed file.
5. Delete `notes_1.txt` and `notes_2.txt` using a single `del` command with wildcards. Capture the final `dir /B` output.

Tasks 6-10: Advanced Operations and Scripting

1. Create a file called `my_data.txt` with at least four lines using `echo` and >>. Then read it with `type my_data.txt` and capture the output.
2. Use `find` to search for a keyword (e.g., ”project”) in `my_data.txt` and pipe the results to `clip`. Paste the results into Notepad and capture a screenshot.
3. List all files in the `Projects` folder and redirect the output to `projects_list.txt`. Open it in Notepad and capture a screenshot of the file.
4. Create a temporary `doskey` alias called `mydir` that runs `dir /B`, test it, and capture the output. Then explain what would be required to make it persistent.

5. Run `help dir` and redirect the output to `help_output.txt`. Open the file in Notepad and capture a screenshot showing at least the first page of help content.

Grading Rubric

Section	Questions	Points Each	Total
Multiple Choice	20	1	20
Short Answer	10	2	20
Hands-On Tasks	10	3	30
Total	**40**	-	**70**

Passing Score: 49 points (70%)

Helpful Resources for Review

- CMD Command Reference
- Navigation and File System
- Using Pipes and Filtering
- DOSKEY and Aliases
- Screen Reader Accessibility Tips

Submission Checklist

☐ All 20 multiple choice questions answered
☐ All 10 short answer questions answered (1-2 sentences each)
☐ All 10 hands-on tasks completed with evidence captured
☐ Files/screenshots organized and labeled clearly
☐ Submission includes this checklist

Screen Reader Accessibility Guide for Windows Command Prompt (CMD)

Target Users: NVDA, JAWS, and other screen reader users
Last Updated: 2026

This guide supports the CMD Foundation curriculum and helps screen reader users navigate and work efficiently with the Windows Command Prompt.

Table of Contents

Getting Started with Screen Readers

Which Screen Reader Should I Use? Both NVDA and JAWS work with CMD; NVDA is free and often easiest to start with. JAWS provides advanced features for power users. Dolphin SuperNova and Windows Narrator are also options:

- Dolphin SuperNova: commercial speech, braille, and magnification (use vendor docs for key mappings).
- Windows Narrator: built into Windows for quick, no-install access.

Before You Start

1. Start your screen reader before opening CMD.
2. Open Command Prompt and listen for the window title and prompt.
3. If silent, press Alt+Tab to find the window.

What is CMD? CMD (Command Prompt) is the classic Windows shell. Common commands include `dir`, `type`, and `more`. Paths use backslashes (e.g., `C:\Users\You`).

NVDA-Specific Tips

NVDA is available from https://www.nvaccess.org/

Dolphin SuperNova Dolphin SuperNova: https://yourdolphin.com/supernova/ — commercial screen reader and magnifier; check institutional licensing.

Windows Narrator Windows Narrator: `https://support.microsoft.com/narrator` — built-in, simpler command set; useful when third-party readers are unavailable.

Key Commands for CMD

Command	What It Does
NVDA+Home	Read the current line (your command or output)
NVDA+Down Arrow	Read from cursor to end of screen
NVDA+Up Arrow	Read from top to cursor
NVDA+Page Down	Read next page
NVDA+Page Up	Read previous page
NVDA+F7	Open the Review Mode viewer (can scroll through text)

Example: Reading Long Output If `dir` produces many lines, redirect to a file and open it in Notepad:

```
dir /b > list.txt
notepad list.txt
```

`dir /b` shows one item per line (screen reader friendly).

JAWS-Specific Tips

JAWS is available from https://www.freedomscientific.com/

Key Commands for CMD

Command	What It Does
Insert+Down Arrow	Read line by line downward
Insert+Up Arrow	Read line by line upward
Insert+Page Down	Read next page of text
Insert+Page Up	Read previous page of text

Example: Reading Long Output

1. Redirect: `dir /b > list.txt`
2. Open Notepad: `notepad list.txt`
3. Use Insert+Ctrl+Down to read full contents.

General Terminal Accessibility

Understanding the CMD Layout The Command Prompt window shows a title bar, a content area, and the prompt (e.g., `C:\Users\YourName>`).

The CMD Prompt Example:

```
C:\Users\YourName>
```

Navigation Sequence

1. Screen reader announces the title
2. Then it announces the prompt line
3. Anything above prompt is prior output

Working with Long Output

Solution 1: Redirect to a File

```
dir /b > list.txt
notepad list.txt
```

Solution 2: Use Pagination

```
type largefile.txt | more
```

Use Space for next page and Q to quit.

Solution 3: Filter Output

```
dir /b | findstr /R "\.scad$"
```

Solution 4: Count Before Displaying

```
dir /b | find /v "" /c
```

Keyboard Shortcuts Reference

Key	Action
Up Arrow	Show previous command
Down Arrow	Show next command
Tab	Auto-complete file/folder names
Home	Jump to start of line
End	Jump to end of line
Ctrl+C	Stop command
Enter	Run command

Troubleshooting

Problem: ”I Can't Hear the Output”

1. Redirect to file and open in Notepad.
2. Use End to jump to the end of text.

Problem: ”Tab Completion Isn't Working”

1. Type at least one character before Tab.

Problem: ”Command Not Found”

1. Use `where programname` to find installed programs.

Pro Tips

1. Use `dir /b` for one-per-line listings.
2. Create a personal notes file and open it in Notepad for quick reference.

Recommended Workflow

1. `cd` to the project folder
2. `dir /b` to list files
3. Redirect large output to files and open in Notepad

Additional Resources

- NVDA Documentation: https://www.nvaccess.org/documentation/
- JAWS Documentation: https://www.freedomscientific.com/support/
- Windows CMD Reference: https://docs.microsoft.com/windows-server/administration/windows-commands/windows-commands
- NVDA Documentation: https://www.nvaccess.org/documentation/
- JAWS Documentation: https://www.freedomscientific.com/support/
- Dolphin SuperNova: https://yourdolphin.com/supernova/
- Windows Narrator: https://support.microsoft.com/narrator

Git Bash

This section covers terminal fundamentals, screen reader accessibility, and command-line basics needed before diving into 3D design with OpenSCAD.

Time commitment: ~10 hours
Skills gained: Terminal navigation, file operations, basic scripting, keyboard-only workflow mastery

Note

This curriculum uses Git Bash - a free Unix-style terminal for Windows that comes bundled with Git for Windows https://git-scm.com/. It can be installed at the provided link or elde via `winget` in the terminal by typing `winget install Git.Git`. All commands in this section use standard bash syntax, which also works on Linux and macOS.

Git Bash Curriculum Overview

Total Duration: 20-25 hours (for screen reader users)
Target Audience: Screen reader users, accessibility-first learners

Prerequisites: Basic Windows computer familiarity

Note: Time estimates reflect the additional time needed for screen reader navigation, text-to-speech processing, and careful keyboard-based workflows.

What is Git Bash?

Git Bash is a Unix/bash shell that runs natively on Windows. It provides the same powerful command-line tools as macOS and Linux while keeping you on Windows.

Why learn Git Bash?

- Cross-platform compatibility: Same commands work on Windows, macOS, and Linux
- Professional standard: Developers worldwide use bash
- Accessibility: Git Bash is 100% screen reader compatible with NVDA and JAWS
- Powerful: Access to Unix tools that Windows Command Prompt and PowerShell can't match
- 3D printing workflows: Essential for OpenSCAD scripting and automation
- Future-proof: Learning bash makes you adaptable to any computer system

Curriculum Structure

Part 1 (Intro): Comparing Command Line Interfaces Start here if you're new to terminals:

- Command Line Interface Selection Guide - Decision guide comparing PowerShell, CMD, and Git Bash
- Learn which option fits your learning style
- Understand when to use each tool

Core Curriculum (11 Lessons)

The main curriculum teaches you to use Git Bash for real work.

Foundation Level (Lessons Pre - 1)

GitBash-Pre: Your First Terminal (1.5-2.5 hours) What you'll learn:

- How to open and close Git Bash
- Understanding the prompt and what it tells you
- Your first commands: `pwd`, `ls`, `cd`
- How to navigate folders safely
- Screen reader accessibility features

Key Skills:

- Know your current location (`pwd`)
- List what's around you (`ls`)
- Move to different folders (`cd`)
- Use Tab completion (Game-changer for screen reader users!)
- Redirect output to files for easier reading

Hands-On Work:

- Open Git Bash 5 times from different methods
- Navigate to 10 different locations
- List contents in 5 different folders
- Create your first test file
- Read it back with a screen reader

GitBash-0: Getting Started - Layout, Paths, and the Shell (1-1.5 hours)
What you'll learn:

- Unix-style paths vs Windows paths
- Absolute vs relative paths
- Understanding the prompt
- Path shortcuts: `~`, `.`, `..`
- Windows drive letters in bash (`/c/` for C:)

Key Skills:

- Convert Windows paths to Git Bash format
- Navigate using both absolute and relative paths
- Understand path structure: `/c/Users/Name/Documents`
- Recognize shortcuts in paths

Hands-On Work:

- Navigate to 5 Windows locations using bash equivalents
- Create a map of paths you use frequently
- Practice relative paths in a deep folder structure
- Document your project folder paths

GitBash-1: Navigation - Moving Around Your File System (1.5-2.5 hours)
What you'll learn:

- Mastering `cd` for folder navigation
- Tab completion strategies
- Going up levels with `..`
- Creating folder structures
- Using shortcuts efficiently

Key Skills:

- Navigate confidently to any location
- Use Tab completion without looking

- Create and organize folder structures
- Jump between frequently-used locations

Hands-On Work:

- Create a 5-level folder structure and navigate it
- Navigate 20+ different locations
- Build a ”favorites” list of important paths
- Time yourself: navigate blindly using only commands

Intermediate Level (Lessons 2 - 4)

GitBash-2: File and Folder Manipulation (2-2.5 hours) What you'll learn:

- Creating files and folders: `mkdir`, `touch`, `echo > file`
- Copying files: `cp`
- Moving and renaming: `mv`
- Deleting safely: `rm` (with caution!)
- Listing with details: `ls -l`
- Checking file sizes: `du`, `wc`

Key Skills:

- Create organized folder structures
- Copy files to new locations
- Rename files safely
- Delete files and folders
- Understand file properties and sizes

Hands-On Work:

- Create 20+ files and organize into folders
- Copy entire folder structures
- Rename 10 files at once
- Practice safe deletion with confirmation
- Build a file organization system for 3D printing projects

GitBash-3: Input, Output & Piping (2-2.5 hours) What you'll learn:

- Redirecting output: >, >>
- Reading files: `cat`, `less`
- Searching text: `grep`
- Sorting data: `sort`
- Counting lines: `wc`
- Piping commands together: |
- Combining tools for powerful workflows

Key Skills:

- Save command output to files

- Search through text efficiently
- Sort and organize data
- Combine multiple commands into workflows
- Debug by redirecting output to files

Hands-On Work:

- Create data files and search through them
- Sort lists of file names and information
- Pipe 10+ different command combinations
- Create scripts that redirect output to files
- Build a file-processing workflow

GitBash-4: Environment Variables & Aliases (1.5-2 hours) What you'll learn:

- Understanding environment variables: `echo $PATH`, `echo $HOME`
- Setting variables: `export MYVAR="value"`
- Creating aliases: `alias ll="ls -la"`
- Editing `.bashrc` configuration file
- Making changes permanent

Key Skills:

- Understand the `$PATH` and why it matters
- Create shortcuts for long commands
- Set up your shell environment
- Make your configuration permanent

Hands-On Work:

- View all your environment variables
- Create 5 useful aliases
- Modify your `.bashrc` file
- Test variables from different locations
- Build a custom shell environment

Advanced Level (Lessons 5 - 6)

GitBash-5: Filling in the Gaps - Shell Profiles, History, and Debugging (2-2.5 hours) What you'll learn:

- Using command history: `history`, `!`, reverse search
- Debugging commands with `set -x`
- Understanding `.bashrc` vs `.bashprofile`
- Viewing and modifying history
- Finding and fixing mistakes

Key Skills:

- Reuse previous commands efficiently
- Debug scripts and workflows
- Configure shell startup behavior
- Learn from your command history

Hands-On Work:

- Review your command history
- Create a custom history system
- Debug 5 failing commands
- Set up a personalized `.bashrc`
- Create aliases for common debugging tasks

GitBash-6: Advanced Terminal Techniques - Scripts, Functions & Professional Workflows (2.5-3.5 hours) What you'll learn:

- Writing bash scripts (`.sh` files)
- Creating functions for reuse
- Using loops: `for`, `while`
- Conditional statements: `if`, `else`
- Professional script structure
- Error handling and logging

Key Skills:

- Write programs in bash
- Create reusable functions
- Automate repetitive tasks
- Handle errors gracefully
- Professional-grade scripts

Hands-On Work:

- Write 5 useful bash scripts
- Create 3 reusable functions
- Build loops that process files
- Write scripts with error checking
- Create a file backup automation system

Comprehensive Assessment (2.5-5 hours)

GitBash Unit Test & Practice What you'll do:

- 30 practical exercises covering all skills
- Real-world scenarios with 3D printing workflows
- Debugging challenges
- Script writing projects
- Performance optimization tasks

Success Criteria:

- Complete all 30 exercises correctly
- Debug 5 failing scripts
- Write 2 advanced automation scripts
- Pass all checkpoint tests

Lesson Progression Map

```
Pre: First Terminal (accessibility focus)
  v
GitBash-0: Paths & Layout (foundations)
  v
GitBash-1: Navigation (mastery)
  v
GitBash-2: Files & Folders (basic manipulation)
  v
GitBash-3: Piping & Redirection (powerful combinations)
  v
GitBash-4: Variables & Aliases (customization)
  v
GitBash-5: History & Debugging (refinement)
  v
GitBash-6: Scripts & Functions (advanced)
  v
Unit Test & Practice (comprehensive)
```

Command Reference by Lesson

GitBash-Pre

- `pwd` - Show current location
- `ls` - List files
- `cd` - Change directory
- `echo "text" >` - Create files
- `cat` - Read files

GitBash-0

- `pwd` - Current location
- `ls` - List contents
- `cd ~` - Go home
- `cd ..` - Go up
- `cd /path` - Go to path

GitBash-1

- `cd FolderName` - Enter folder
- `cd ..` - Go up one level
- `cd ~` - Go home
- `ls` - List contents
- Tab completion (Press Tab)

GitBash-2

- `mkdir FolderName` - Create folder
- `touch FileName` - Create file
- `cp Source Dest` - Copy file
- `mv Source Dest` - Move/rename
- `rm FileName` - Delete file
- `rmdir FolderName` - Delete empty folder
- `ls -l` - List with details

GitBash-3

- `cat FileName` - Display file
- `grep "text" File` - Search
- `sort FileName` - Sort lines
- `wc` - Count words/lines
- `command > file` - Redirect output
- `command | other` - Pipe commands
- `less FileName` - Read page by page

GitBash-4

- `echo $VAR` - Show variable
- `export VAR="value"` - Set variable
- `alias name="command"` - Create shortcut
- `.bashrc` - Edit startup file
- `source ~/.bashrc` - Reload config

GitBash-5

- `history` - Show past commands
- `!CommandNumber` - Rerun command
- `Ctrl+R` - Search history
- `set -x` - Debug mode
- `.bashrc` vs `.bashprofile` - Config files

GitBash-6

- `#!/bin/bash` - Script header
- `function name() {}` - Define function
- `for var in list; do` - Loops
- `if [condition]; then` - Conditionals
- `source script.sh` - Run script
- `$1, $2` - Arguments

Accessibility Features Throughout

Every lesson includes:

- Screen reader-tested examples using NVDA and JAWS
- Text-based output - No graphics, all linear text
- Tab completion strategies - Optimized for speech feedback
- File redirection methods - Save output when needed
- Keyboard-only workflows - No mouse required
- Practical tips for both NVDA and JAWS users

Time Breakdown

Component	Time
GitBash-Pre	1.5-2.5 hours
GitBash-0	1-1.5 hours
GitBash-1	1.5-2.5 hours
GitBash-2	2-2.5 hours
GitBash-3	2-2.5 hours
GitBash-4	1.5-2 hours
GitBash-5	2-2.5 hours
GitBash-6	2.5-3.5 hours
Unit Test	2.5-5 hours
Total	20-25 hours

Learning Approach

Core Teaching Method

1. Understand WHY - Each lesson starts with purpose
2. Learn WHAT - Commands and concepts
3. Practice HOW - Hands-on exercises
4. Master IT - Checkpoint questions and real-world work

Recommended Pace

- Fast track: Complete in 2-3 weeks (2-3 hours daily)
- Standard pace: Complete in 8-12 weeks (2-3 hours, 3x per week)
- Self-paced: Work at your own speed with breaks

Best Practices

- Take breaks between lessons
- Do all practice exercises - don't skip
- Answer checkpoint questions before moving on
- Review previous lessons if stuck

- Use screen reader at full speed once comfortable

Prerequisites & Assumptions

This curriculum assumes you:

- Have Git Bash installed (free from https://git-scm.com)
- Use a screen reader (NVDA, JAWS, or similar)
- Have Windows 10 or later
- Can use a keyboard efficiently
- Have a text editor (Notepad, VS Code, etc.)

What You'll Be Able to Do

After completing this curriculum:

Basic Skills (After Lesson 1)

- Navigate any folder on your computer
- List files to understand what's available
- Create and organize folders
- Find files quickly

Intermediate Skills (After Lesson 4)

- Create, copy, move, and delete files
- Search through text for information
- Combine commands into powerful workflows
- Customize your terminal environment

Advanced Skills (After Lesson 6)

- Write bash programs and scripts
- Automate repetitive tasks
- Build professional tools
- Solve complex problems
- Work efficiently across Windows, macOS, and Linux

Assessment & Completion

Unit Test (Comprehensive)

- 30 practical exercises
- Real-world 3D printing scenarios
- Script writing projects
- Performance: 90%+ correct = mastery

Certification Upon completion:

- Printable certificate of completion
- Skills assessment document
- Portfolio project (optional)

Moving Forward

After completing this curriculum, you can:

Use Git Bash Daily

- Automate 3D printing workflows
- Manage file systems efficiently
- Create custom tools

Learn More

- Advanced bash scripting
- Using Git for version control
- Integrating with OpenSCAD
- Professional shell environments

Transition to Other Systems

- Skills transfer to macOS Terminal
- Skills transfer to Linux systems
- Knowledge applies to cloud environments (AWS, Azure, GCP)

Frequently Asked Questions

Q: Why Git Bash instead of PowerShell or CMD? A: Git Bash gives you the same commands as macOS and Linux, making it a cross-platform skill that's more valuable long-term.

Q: How is this different from regular bash? A: Git Bash is bash running on Windows. 100% compatible with standard bash. All commands work the same way.

Q: Will I need to use a mouse? A: No! The entire curriculum is keyboard-based. Perfect for screen reader users.

Q: Can I take breaks between lessons? A: Yes! Each lesson stands alone. You can take days or weeks between lessons.

Q: What if I get stuck? A: Each lesson has extension problems for deeper learning and troubleshooting guides for common issues.

Q: Will this help with 3D printing? A: Absolutely! Many lessons include 3D printing workflows and OpenSCAD automation examples.

Support & Resources

- NVDA Screen Reader: https://www.nvaccess.org/
- JAWS Screen Reader: https://www.freedomscientific.com/products/software/jaws/
- Bash Manual: https://www.gnu.org/software/bash/manual/
- Git Bash Documentation: https://git-scm.com/book/
- Linux Command Reference: https://linuxcommand.org/

Curriculum Philosophy

This curriculum is built on:

- Accessibility First - Every lesson designed for screen readers
- Practical Learning - Real commands, real workflows
- Progressive Complexity - Build skills systematically
- Hands-On Practice - Learn by doing
- Professional Standards - Industry-ready skills

Ready to Begin?

Start with the Command Line Interface Selection Guide to see why Git Bash might be the right choice for you.

Then proceed to GitBash-Pre: Your First Terminal to begin your journey!

Goal: You can navigate to any folder and see what's in it with your screen reader.

Phase 2: Intermediate User -> Power User

Lesson	Duration	What You'll Learn
GB-2: File & Folder Manipulation	60 min	Create, copy, move, delete files/folders
GB-3: Input, Output & Piping	60 min	Chain commands together, redirect output
GB-4: Environment Variables & Aliases	45 min	Automate settings, create shortcuts
GB-5: Filling in the Gaps	45 min	Profiles, history, debugging

Goal: You can create folders, manage files, and combine commands to accomplish complex tasks.

Phase 3: Professional Skills (Beyond Curriculum)

Topic	When to Learn
Shell scripting (.sh files)	After GB-5
Functions & Loops	After GB-5
Error Handling	After GB-5
3D Printing Integration	After all above

How to Use This Curriculum

If You've Never Used a Terminal Before

1. Read Screen Reader Accessibility Guide completely
2. Do GB-Pre: Your First Terminal exercises
3. Continue with GB-0, GB-1, etc.

If You've Used a Terminal Before (But Not with a Screen Reader)

1. Skim Screen Reader Accessibility Guide
2. Quickly review GB-Pre
3. Move to GB-0 for deeper learning

If You're Experienced with Terminal + Screen Reader

1. Jump to specific lessons you need (GB-2, GB-3, etc.)
2. Use the Quick Reference sections
3. Skip the practice exercises, do the quizzes to verify knowledge

How Each Lesson is Structured

1. Learning Objectives - What you'll be able to do
2. Key Commands - The important ones to memorize
3. Step-by-Step Examples - How to actually do it
4. Practice Exercises - Hands-on work
5. Quiz Questions - Check your understanding
6. Extension Problems - Go deeper if interested

Screen Reader Tips Throughout the Curriculum

Every lesson includes:

- [SR] symbols marking screen reader-specific sections
- Tips for NVDA and JAWS users separately
- Solutions for common accessibility issues
- Workarounds for long outputs

Quick Start Guide (First 15 Minutes)

1. Open Git Bash (Start menu -> type Git Bash -> Enter)
2. Run these commands:

```
pwd
ls
cd Documents
pwd
```

3. See how your screen reader reads each output
4. Try Tab completion: type `cd D` and press Tab
5. Create a file:

```
echo "I am learning Git Bash" > learning.txt
cat learning.txt
```

Important Rules

Rule 1: Always Know Where You Are

```
pwd
```

Rule 2: Check Before You Delete

```
ls
```

Rule 3: Use `ls` for listings

```
ls
# or for one-per-line (screen reader friendly):
ls -1
```

Rule 4: When Lost, Redirect to a File

```
command-name > output.txt
notepad output.txt
```

Rule 5: Save Everything You Create

```
mkdir my-practice-folder
cd my-practice-folder
```

Troubleshooting: ”Nothing Works!”

1. Can't hear Git Bash? Make sure screen reader is running BEFORE Git Bash. Try Alt+Tab.
2. Commands not working? Check spelling. Make sure you pressed Enter.
3. Can't read the output? Redirect to file: `command > output.txt`, then `notepad output.txt`
4. Something ran forever? Press Ctrl+C to stop it.
5. Completely confused? Go back to GB-Pre and start over.

Curriculum Map

```
START HERE v
+---- Screen Reader Accessibility Guide (reference throughout)
+---- GB-Pre: Your First Terminal (absolute beginner entry point)
+---- GB-0: Getting Started (paths & navigation foundations)
+---- GB-1: Navigation (comfortable moving around)
+---- GB-2: File & Folder Manipulation (create/move/delete)
+---- GB-3: Input, Output & Piping (chain commands)
+---- GB-4: Environment Variables & Aliases (automation)
+---- GB-5: Filling in the Gaps (profiles & history)
+---- GBUnitTest (comprehensive practice & assessment)
       v
   NEXT: 3D Printing Integration Lessons
       v
   ADVANCED: Bash Scripting
```

Questions? Stuck? Refer back to this page and the Screen Reader Accessibility Guide.

Now open GB-Pre and let's get started!

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

Git Bash Introduction - Screen-Reader Friendly (JAWS/NVDA)

Estimated time: 20-30 minutes

Target audience: Users who are new to the command line and use screen readers.

Learning Goals

- Open and identify the Git Bash prompt
- Run basic commands: `pwd`, `ls -1`, `cd`, `touch`, `cat`, `echo`
- Redirect and page through long outputs for screen readers
- Use `--help` and `which` to discover more about commands

Starting Git Bash

Windows

1. Press Windows, type `Git Bash`, then press Enter.
2. Git Bash opens a terminal window. You'll hear the prompt announced by your screen reader.

When Git Bash starts you will hear the prompt and usually your current path. The default prompt looks something like:

```
YourName@COMPUTER MINGW64 ~
$
```

Essential Commands - Short List

- `pwd` - print working directory (where you are now)
- `ls -1` - list file and folder names one per line (recommended for screen readers)
- `cd <path>` - change directory
- `touch <filename>` - create an empty file
- `cat <file>` - show file contents
- `echo <text>` - print text

Quick examples:

```
pwd
ls -1
cd Documents
touch notes.txt
echo "hello" > notes.txt
cat notes.txt
```

Paging Long Outputs for Screen Readers

- If an output is long, redirect to a file and open it in an editor:

```
some-command > output.txt
notepad output.txt
```

- Or pipe to `less`: `some-command | less` (press Space to advance, Q to quit).
- For very long listings: `ls -1 > list.txt && notepad list.txt`

Help and Discovery

- `command --help` shows help: `ls --help` or `grep --help`
- `man command` shows the manual page (pipe it to a file for easy reading): `man ls > lshelp.txt && notepad lshelp.txt`
- `which <name>` locates a program: `which openscad`

Running Scripts and Executables

- Make a script executable: `chmod +x script.sh`
- Run a script in the current directory: `./script.sh`
- Run an executable by full path: `'/c/Program Files/OpenSCAD/openscad.exe'`

Screen Reader Specific Tips (JAWS, NVDA)

- Use `ls -1` to avoid cluttered, multi-column listings
- Redirect large outputs to a text file and open it in an editor your screen reader handles well
- If output stops being announced, try moving focus with Alt+Tab and back, or briefly pressing Ctrl or Shift to force announcement

Git Bash vs Windows Command Prompt

Task	Git Bash	Command Prompt
List files	`ls -1`	`dir`
Show current path	`pwd`	`cd`
Read a file	`cat file.txt`	`type file.txt`
Create file	`touch file.txt`	`type nul > file.txt`
Copy file	`cp src dest`	`copy src dest`
Move/rename	`mv old new`	`move old new`
Delete file	`rm file.txt`	`del file.txt`

Git Bash uses Unix commands that also work on Linux and macOS.

Suggested Learning Path

1. Open Git Bash, run `pwd`, `ls -1`, `cd` into Documents.
2. Create a file with `touch test.txt`, add a line with `echo "hi" > test.txt`, and read it with `cat test.txt`.
3. Use `ls --help` and redirect output to a file to read comfortably.

Hands-On: Explore the Terminal (Beginner)

1. Run `pwd` and speak or note the output.
2. Run `ls -1` in your home directory. If the list is long, save it: `ls -1 > listing.txt && notepad listing.txt`
3. Create a folder and file: `mkdir GBPractice && cd GBPractice && touch notes.txt`
4. Add a line: `echo "My first note" > notes.txt && cat notes.txt`

Extension Exercises

1. Use wildcards to list only `.scad` files: `ls *.scad`. Proceed to search for other types of files.
2. Save a long command's output and search it with `grep`.
3. Create a small script (`chmod +x`, then `./`) that prints a timestamp and test it.
4. Build a project template generator: write a script that creates a folder structure for a new 3D printing project.
5. Create a personalized `.bashrc` with aliases for frequently used commands; test it after restarting Git Bash.
6. Write a Bash script that combines multiple concepts: navigates folders, creates files, and performs file operations.
7. Create a screen-reader navigation guide for Git Bash; test with an actual screen reader if available.

8. Build a data analysis script: read a text or CSV file, filter and process data, and generate a report.
9. Develop a troubleshooting aide: common Git Bash error messages, their causes, and solutions.
10. Design a comprehensive Git Bash learning portfolio: document skills developed, commands mastered, scripts created, and practical applications.

Quick Quiz (10 questions)

1. What command shows your current directory?
2. How do you list names only, one per line, for screen readers?
3. How do you create an empty file?
4. How do you run a script in the current directory?
5. How do you page through long output?
6. True or False: Git Bash commands also work on Linux and macOS.
7. What does the ~ symbol represent in a Git Bash path?
8. Explain why redirecting output to a file (e.g., `command > output.txt`) is helpful for screen reader users.
9. What is the difference between `--help` and `man` in Git Bash?
10. Describe a practical scenario where you would use piping (|) in Git Bash.

References

- Git for Windows. (2024). *Git Bash documentation*. https://gitforwindows.org/
- GNU. (2024). *Bash reference manual*. https://www.gnu.org/software/bash/manual/bash.html
- The Linux Documentation Project. (2024). *Bash Beginners Guide*. https://tldp.org/LDP/Bash-Beginners-Guide/html/

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

Git Bash Tutorial

Estimated time: 30-45 minutes

Learning Objectives

- Launch Git Bash and identify the prompt
- Understand and use basic path notation (`~`, `./`, `../`)
- Use `pwd`, `ls -1`, and `cd` to navigate the filesystem
- Open files in an external editor and run simple commands

Materials

- A computer with Git Bash installed (install via https://git-scm.com/)
- Access to a text editor (Notepad, VS Code)

Pre-Requisite Knowledge

- Typing and basic text-editing skills
- Familiarity with file/folder concepts and basic OS navigation
- Basic screen-reading familiarity (if applicable)

What is Git Bash?

Git Bash is a free terminal application for Windows that provides a Bash (Unix shell) environment. It is installed as part of Git for Windows. When you open Git Bash, you can use Unix commands like `ls`, `cat`, `grep`, and `echo` - the same commands used on Linux and macOS - right on your Windows computer.

Git Bash is useful for:

- Running CLI programs (like OpenSCAD, slicers, or 3DMake)
- Navigating the filesystem with keyboard-only commands
- Automating repetitive tasks with scripts
- Accessibility: works cleanly with NVDA, JAWS, and other screen readers

What We'll Do and Why

You'll use Git Bash to run CLI programs, navigate the filesystem, and manipulate files - tasks that are especially efficient when using a keyboard or a screen reader.

Quick Tutorial & Core Concepts

Paths and Navigation

- `~` - home directory (e.g., `/c/Users/YourName`)
- `.` - current directory
- `..` - parent directory
- `./` - current directory shortcut (used to run scripts: `./script.sh`)
- `../` - parent directory shortcut
- Use Tab to autocomplete files and folders

Git Bash uses forward slashes (/) for paths, just like Linux. Your Windows `C:\Users\YourName` folder is accessible as `/c/Users/YourName` or simply `~`.

Useful Commands (Examples)

```
pwd # show current directory
ls -1 # list files, one per line (screen-reader friendly)
cd path/to/dir # change directory
whoami # current user
```

Wildcards

- `*` matches zero or more characters
- `?` matches a single character

Use `ls *.scad` to filter by extension, for example.

Common Operations

File and Folder Manipulation

```
mkdir my-folder # create folder
mkdir -p a/b/c # create nested folders at once
cp -r src dest # copy (use -r for directories)
mv oldname newname # rename or move
rm file # remove file
rm -r folder # remove folder and contents
touch filename.txt # create new empty file
```

Input, Output, and Piping

```
echo 'hello' | clip # copy to clipboard
command > output.txt # redirect output to file
command >> output.txt # append output to file
command > /dev/null # discard/suppress output
```

Use | to pipe output and > to redirect into files.

Editing and Running Programs

```
notepad file.txt # open in Notepad (Windows)
code file.txt # open in VS Code (if in PATH)
./script.sh # run a script in the current directory
chmod +x script.sh # make a script executable
```

Screen-Reader Friendly Tips

- Prefer `ls -1` for name-only, one-per-line output.
- Filter lists with `grep`: `ls -1 ~/Documents | grep "\.scad$"` (files only ending in .scad).
- Redirect very long outputs to a file and open it in an editor.

Error Handling and Control

- Abort a running command: `Ctrl+C`
- View history: Up/Down arrows or `history`
- Clear screen: `clear` (or `Ctrl+L`)
- If an error is long, read the first few lines for the gist and copy short snippets into a file to examine.

Environment Variables & PATH

Environment variables configure your session. `PATH` tells the shell where to find executables.

```
echo $PATH
echo $HOME
echo $USER
```

To add a directory to PATH for the current session:

```
export PATH="$PATH:/path/to/my/tools"
```

To make it permanent, add that line to `~/.bashrc`.

Running CLI Applications and Archives

To extract ZIP archives in Git Bash:

```
# Unzip a file
unzip file.zip -d destinationfolder

# Or use Windows' built-in tool (from Git Bash):
powershell.exe Expand-Archive -Path file.zip -DestinationPath folder
```

Aliases and Cross-Platform Notes

Git Bash provides Unix commands that work identically on Linux and macOS. This means your Git Bash skills transfer directly if you ever work on a Mac or Linux system.

Useful aliases to add to `~/.bashrc`:

```
alias ll='ls -la'
alias la='ls -1a'
alias ..='cd ..'
alias ...='cd ../..'
```

Step-by-step Tasks (Hands-On)

1. Open Git Bash; listen for the prompt and current path.
2. Run `pwd` to confirm your location.
3. Run `ls -1` in your home directory and note the output.
4. Practice `cd Documents`, `cd ../`, and `cd ~` to move between folders.
5. Create and open a file: `touch example.txt && notepad example.txt` (or `code example.txt`).

Checkpoints

- After step 3 you should be able to state your current directory.
- After step 5 you should be able to create and open a text file from Git Bash.

Quick Quiz (10 questions)

1. What command prints your current directory?
2. What does ~ represent?
3. How do you list only names, one per line?
4. How do you go up one directory level?
5. How would you open `notes.txt` in Notepad from Git Bash?
6. True or False: The pipe operator `|` sends output to a file.
7. Explain why running a script requires `chmod +x` first.
8. What is the difference between running a script with `./script.sh` versus `bash script.sh`?
9. Describe how you would handle a very long command output when using a screen reader.
10. What does the `PATH` environment variable do, and why is it important when running programs like OpenSCAD?

Extension Problems

1. Create a folder `OpenSCADProjects` in Documents and verify its contents.
2. Create three files named `a.scad`, `b.scad`, `c.scad` and list them with a wildcard.
3. Save `ls -1 ~/Documents` output to `doclist.txt` and open it.
4. Try tab-completion in a deeply nested folder and note behavior.
5. Capture `pwd` output into a file and open it: `pwd > cwd.txt && notepad cwd.txt`.
6. Build an automated setup script that creates a complete project directory structure, initializes placeholder files, and generates a README.
7. Create a Git Bash cheat sheet for your most-used commands; organize by category (navigation, files, scripting, troubleshooting).
8. Write a non-visual tutorial for Git Bash basics; use audio descriptions and keyboard-only navigation as the primary learning method.
9. Develop a workflow automation script: combines multiple Git Bash concepts (folders, aliases, piping) to solve a real 3D printing task.
10. Create a Git Bash proficiency self-assessment: list all concepts covered, provide test commands for each, and reflect on what you learned.

References

- Git for Windows. (2024). *Git Bash*. https://gitforwindows.org/
- GNU. (2024). *Bash reference manual*. https://www.gnu.org/software/bash/manual/bash.html
- The Linux Documentation Project. (2024). *Bash Beginners Guide*. https://tldp.org/LDP/Bash-Beginners-Guide/html/

Helpful Resources

- Git for Windows
- Bash Reference Manual (GNU)
- Bash Beginners Guide
- GNU Coreutils

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

Screen Reader Accessibility Guide for Git Bash

Target Users: NVDA, JAWS, and other screen reader users
Last Updated: 2026

This guide is used throughout the Git Bash Foundation curriculum to help screen reader users navigate and work efficiently with the terminal.

Table of Contents

Getting Started with Screen Readers

Which Screen Reader Should I Use? Both NVDA and JAWS work well with Git Bash, but they have different strengths:

Feature	NVDA	JAWS
Cost	Free	Commercial (paid)
Installation	Simple	Complex but thorough
Git Bash Support	Excellent	Excellent
Learning Curve	Gentle	Steeper
Customization	Good	Extensive

Recommendation: Start with NVDA if you're new to screen readers. Both will work for this curriculum.

Additional options: Dolphin SuperNova is a commercial screen reader/ magnifier popular in some education settings; Windows Narrator is built into Windows and useful for quick access without installing additional software. Choose based on availability and personal preference.

Before You Start

1. Make sure your screen reader is running before opening Git Bash
2. Open Git Bash and let your screen reader read the window title and prompt
3. If you don't hear anything, press Alt+Tab to cycle windows and find Git Bash
4. Use your screen reader's screen review features to understand the layout

What is Git Bash? Git Bash is a terminal application for Windows that provides a Unix-style command-line experience (Bash shell). It is installed as part of Git for Windows, which is free software available at https://git-scm.com/. When you open Git Bash, you get the same `bash`, `ls`, `grep`, `cat`, and other Unix tools used on Linux and macOS - but running on your Windows computer.

NVDA-Specific Tips

NVDA is free and available from https://www.nvaccess.org/

Key Commands for Git Bash

Command	What It Does
NVDA+Home	Read the current line (your command or output)
NVDA+Down Arrow	Read from cursor to end of screen
NVDA+Up Arrow	Read from top to cursor
NVDA+Page Down	Read next page
NVDA+Page Up	Read previous page
NVDA+F7	Open the Review Mode viewer (can scroll through text)
NVDA+Shift+Right Arrow	Read next word
NVDA+Shift+Down Arrow	Read entire screen
NVDA+End	Jump to end of line
NVDA+Home	Jump to start of line

Example: Reading Long Output Scenario: You ran `ls` and it listed `50` files.

Solution with NVDA:

1. After the command finishes, press NVDA+Home to read the current line

2. Press NVDA+Down Arrow repeatedly to read all output
3. Or press NVDA+F7 to open Review Mode and use arrow keys to scroll

Tip: Use `ls -1` for Screen Reader Friendly Listings By default, `ls` in Git Bash may display files in columns. Use `ls -1` (the number one, not the letter L) to display one file per line - much easier to follow with a screen reader.

JAWS-Specific Tips

JAWS is a commercial screen reader available from https://www.freedomscientific.com/

Dolphin SuperNova Dolphin SuperNova (commercial): `https://example.com` — provides speech and braille support and tightly integrated magnification. Refer to vendor docs for keyboard mappings; many users rely on review-mode features similar to NVDA/JAWS.

Windows Narrator Windows Narrator (built-in): https://example.com — good for quick checks and lightweight usage; Narrator command keys differ by Windows version (use Narrator key + arrow keys to read). Add Narrator to your workflow when a full third-party screen reader isn't available.

Key Commands for Git Bash

Command	What It Does
Insert+Down Arrow	Read line by line downward
Insert+Up Arrow	Read line by line upward
Insert+Page Down	Read next page of text
Insert+Page Up	Read previous page of text
Insert+End	Jump to end of text on screen
Insert+Home	Jump to start of text on screen
Insert+Ctrl+Down	Read to end of screen
Insert+Ctrl+Up	Read to beginning of screen
Insert+F3	Open JAWS menu

Example: Reading Long Output Scenario: You ran `ls > list.txt` and saved output to a file.

Solution with JAWS:

1. Open the file: `notepad list.txt`
2. In Notepad, press Insert+Ctrl+Down to hear all content
3. Use Insert+Down Arrow to read line by line at your own pace

General Terminal Accessibility

Understanding the Git Bash Layout The Git Bash window contains:

1. Title bar: Window name (e.g., ”MINGW64:/c/Users/YourName”)
2. Content area: Command history and output
3. Prompt: Where you type (e.g., `YourName@COMPUTER MINGW64 ~$`)

Your screen reader reads from top to bottom, but focus is at the prompt (bottom).

The Git Bash Prompt The default Git Bash prompt looks like:

```
YourName@COMPUTERNAME MINGW64 ~/Documents
$
```

- `YourName` = your Windows username
- `COMPUTERNAME` = your computer's name
- `MINGW64` = the environment type (64-bit)
- `~/Documents` = your current location
- `$` = ready for input

Note: Paths in Git Bash use forward slashes (/) and your Windows `C:\Users\YourName` folder appears as `/c/Users/YourName` or simply `~`.

Navigation Sequence When you open Git Bash:

1. Your screen reader announces the window title
2. Then it announces the prompt line
3. Anything before the prompt is previous output
4. Anything after the prompt is where new output will appear

Reading Output Effectively Strategy 1: Immediate Output (Small Amount)

- Run a command
- Your screen reader announces output immediately
- This works well for short outputs (a few lines)

Strategy 2: Large Output (Many Lines)

- Redirect to a file: `command > output.txt`
- Open the file: `notepad output.txt`
- Read in Notepad (easier for long text)

Strategy 3: Searching Output

- Use `grep` to filter:

  ```
  ls | grep "pattern"
  ```

- Only results matching ”pattern” are shown

Working with Long Output

Solution 1: Redirect to a File

```
ls -1 > list.txt
notepad list.txt
```

Solution 2: Use Pagination

```
ls | less
```

- Press Space to see next page
- Press Q to quit
- Note: Some screen readers struggle with `less`, so Solution 1 is preferred

Solution 3: Filter Output

```
ls | grep "\.scad"
```

Solution 4: Count Before Displaying

```
ls | wc -l
```

Tells you how many items there are. If the count is over 20, use the file method.

Keyboard Shortcuts Reference

All Users (Works in Git Bash Regardless of Screen Reader)

Key	Action
Up Arrow	Show previous command
Down Arrow	Show next command
Tab	Auto-complete file/folder names
Shift+Tab	Cycle backward through completions
Home	Jump to start of line
End	Jump to end of line
Ctrl+A	Jump to start of line (alternate)
Ctrl+E	Jump to end of line (alternate)
Ctrl+C	Stop command
Ctrl+L	Clear screen
Enter	Run command
Ctrl+R	Search command history interactively

Troubleshooting

Problem 1: ”I Can't Hear the Output After Running a Command”

1. Redirect to file: `command > output.txt` then `notepad output.txt`
2. Press End or Ctrl+End to go to end of text
3. Use Up Arrow to review previous command

Problem 2: "Tab Completion Isn't Working"

1. Need at least one character - type `cd D` then Tab (not just `cd` then Tab)
2. Check if item exists - use `ls` first to see available items
3. Multiple matches - press Tab twice to list all options

Problem 3: "'Command Not Found' Error"

1. Check spelling carefully
2. Git Bash uses Unix commands: use `ls` not `dir`, use `cat` not `type`
3. Verify the program is installed and in your PATH: `which openscad`

Problem 4: "Paths Look Weird" Git Bash converts Windows paths to Unix style:

- Windows: `C:\Users\YourName\Documents`
- Git Bash: `/c/Users/YourName/Documents` or `~/Documents`

Both refer to the same location. When opening files in Windows apps (like Notepad), you may need to use the Windows path format.

Problem 5: "Command Runs Forever and Won't Stop" Press Ctrl+C

Problem 6: "I Need to Edit My Last Command"

1. Press Up Arrow to show previous command
2. Use arrow keys to move through it
3. Edit the command
4. Press Enter to run the modified version

Pro Tips for Efficiency

1. Use `ls -1` for Screen Reader Friendly Listings

```
ls -1
```

One file per line - much easier to follow with a screen reader.

2. Create Aliases for Frequently Used Commands

```
alias la='ls -1a'
```

Add this to your `~/.bashrc` file to make it permanent.

3. Use Command History Effectively

```
history
```

Run a previous command by number:

```
!5
```

(Runs the 5th command in history)

Or search history interactively:

```
# Press Ctrl+R then type part of a previous command
```

4. Redirect Everything to Files for Accessibility

```
command-name > results.txt
notepad results.txt
```

5. Create a README for Yourself

```
echo "ls -1 means list files one per line (screen reader friendly)" > my-
    notes.txt
echo "cd means change directory" >> my-notes.txt
notepad my-notes.txt
```

Recommended Workflow

For every new task:

1. Know where you are: `pwd`
2. See what's around: `ls -1`
3. Plan your next step: Think before typing
4. Run the command: Type and press Enter
5. Check the output: Use screen reader or redirect to file
6. Move forward: Next command or `cd` to next folder

Quick Reference Card

```
EVERY COMMAND STARTS WITH:
1. pwd (where am I?)
2. ls -1 (what's here?)
3. cd path (go there)

LONG OUTPUT?
-> command > file.txt
-> notepad file.txt

STUCK?
-> Ctrl+C

WANT TO REPEAT?
-> Up Arrow
-> history

NEED HELP?
-> man command-name (or: command --help)
```

Additional Resources

- NVDA Documentation: https://www.nvaccess.org/documentation/
- JAWS Documentation: https://www.freedomscientific.com/support/
- Git for Windows (includes Git Bash): https://git-scm.com/
- Bash Manual: https://www.gnu.org/software/bash/manual/

GitBash-Pre: Your First Terminal - Screen Reader Navigation Fundamentals

Duration: 1.5-2 hours (for screen reader users)
Prerequisites: None - this is the starting point
Accessibility Note: This lesson is designed specifically for screen reader users (NVDA, JAWS)

Important

Git Bash uses Unix/bash commands, different from Windows CMD/PowerShell

What is Git Bash?

Git Bash is a Unix/Linux shell running on Windows. It gives you bash commands (same as macOS and Linux) while staying on your Windows computer.

Why learn this?

- Faster and more precise work (especially for 3D printing scripts and automation)
- Essential for programming and using tools like OpenSCAD
- Accessibility: Git Bash works perfectly with screen readers
- Transferable: Same commands work on macOS and Linux
- Industry-standard: Professional developers use this everywhere

Installing and Opening Git Bash

Installation (If Not Already Installed)

1. Download from: https://git-scm.com/download/win
2. Run the installer
3. Accept defaults (all options work fine)
4. Installation takes ~5 minutes

Opening Git Bash Method 1: Search (Easiest)

1. Press the Windows key alone
2. You should hear "Search"
3. Type: `Git Bash`
4. You'll hear search results appear
5. Press Enter to open

Method 2: From File Explorer

1. Open File Explorer
2. Navigate to any folder
3. Right-click on the folder
4. Look for "Git Bash Here"
5. Press Enter

First Connection: Understanding the Prompt When Git Bash opens, your screen reader will announce the window title and then the prompt.

What you'll hear (typical example):

```
user@computer MINGW64 ~
$
```

What this means:

- `user` = Your username
- `@computer` = Your computer name
- `MINGW64` = Version of Git Bash
- `~` = Your current location (home directory)
- `$` = The prompt is ready for your input

Important

Your cursor is blinking right after the `$`. This is where you type.

Your First Commands (Screen Reader Edition)

Command 1: "Where Am I?" - `pwd` What it does: Tells you your current location

Type this:

```
pwd
```

Press Enter

What you'll hear: Your screen reader will announce the current path, something like:

```
/c/Users/YourName
```

Understanding paths in Git Bash:

- Paths use forward slashes: `/c/Users/YourName/Documents`
- `/c/` means the C: drive (Windows drive)
- This is Unix-style, same as macOS and Linux
- Think of it like folders inside folders: `/c/` (main drive) -> `Users` -> `YourName` -> `Documents`

Command 2: "What's Here?" - `ls` What it does: Lists all files and folders in your current location

Type this:

```
ls
```

Press Enter

What you'll hear: Your screen reader will announce a list of file and folder names, one per line. Perfect for screen readers!

Command 3: "Go There" - `cd Documents` What it does: Changes your location (navigates to a folder)

Type this:

```
cd Documents
```

Press Enter

What you'll hear: The prompt changes to show your new location. You might hear something like:

```
user@computer MINGW64 ~/Documents
$
```

Practice navigation:

1. Run `pwd` to confirm you're in Documents
2. Run `ls` to see what files are in Documents
3. Try going back: `cd ..` (the `..` means "go up one level")
4. Run `pwd` again to confirm
5. Go back to Documents: `cd Documents`

Creating and Viewing Files

Create a Simple File Type this:

```
echo "Hello, Git Bash!" > hello.txt
```

What this does:

- `echo` sends text to the screen (or file)
- `"Hello, Git Bash!"` is the text
- > redirects it to a file called `hello.txt`

Read the File Back Type this:

```
cat hello.txt
```

What you'll hear: Your screen reader announces:

```
Hello, Git Bash!
```

Open and Edit the File Type this:

```
notepad.exe hello.txt
```

This opens the file in Notepad where you can edit it with your screen reader.

Essential Keyboard Shortcuts

These work in Git Bash and are crucial for screen reader users:

Key Combination	What It Does
Up Arrow	Shows your previous command (press again to go further back)
Down Arrow	Shows your next command (if you went back)
Tab	Auto-completes folder/file names
Ctrl+C	Stops a running command
Ctrl+L	Clears the screen
Ctrl+A	Go to beginning of line
Ctrl+E	Go to end of line
Enter	Runs the command

Screen reader tip: These all work perfectly with your screen reader. Try them!

Screen Reader-Specific Tips

NVDA Users

1. Reading Command Output:
   - Use NVDA+Home to read the current line
   - Use NVDA+Down Arrow to read to the end of the screen
   - Use NVDA+Page Down to read the next page
2. Reviewing Text:
   - Use NVDA+Shift+Page Up to review text above

JAWS Users

1. Reading Output:
   - Use Insert+Down Arrow to read line-by-line
   - Use Insert+Page Down to read by page
   - Use Insert+End to jump to the end of text
2. Reading All Text:
   - Use Insert+Down Arrow repeatedly

- Or use Insert+Ctrl+Down to read to the end

Common Issue: ”I Can't Hear the Output” Problem: You run a command but don't hear the output

Solutions:

1. Make sure your cursor is at the prompt (try pressing End or Ctrl+E)
2. Use Up Arrow to go back to your previous command and review it
3. Try redirecting to a file: `command > output.txt` then open the file
4. In NVDA: Try pressing NVDA+F7 to open the Review Mode viewer

Practice Exercises

Complete these in order. Take your time with each one:

Exercise 1: Basic Navigation

1. Open Git Bash
2. Run `pwd` and note your location
3. Run `ls` and listen to what's there
4. Try `cd Documents` or another folder
5. Run `pwd` to confirm your new location
6. Run `ls` in this new location

Goal: You should be comfortable knowing where you are and what's around you

Exercise 2: Using Tab Completion

1. In your home directory, type `cd D` (just the letter D)
2. Press Tab
3. Git Bash should auto-complete to a folder starting with D
4. Repeat with other folder names
5. Try typing a longer name: `cd Down` and Tab

Goal: Tab completion should feel natural

Exercise 3: Creating and Viewing Files

1. Create a file: `echo "Test content" > test.txt`
2. View it: `cat test.txt`
3. Create another: `echo "Line 2" > another.txt`
4. List both: `ls *.txt`

Goal: You understand create, view, and list operations

Exercise 4: Going Up Levels

1. Navigate into several folders: `cd Documents`, then `cd folder1`, etc.
2. From deep inside, use `cd ..` multiple times to go back up
3. After each `cd ..`, run `pwd` to confirm your location

Goal: You understand relative navigation with `..`

Exercise 5: Redirecting Output

1. Create a list: `ls > directorylist.txt`
2. Open it: `notepad.exe directorylist.txt`
3. Read it with your screen reader
4. Close Notepad
5. Verify the file exists: `ls | grep directory`

Goal: You can save long outputs to files for easier reading

Checkpoint Questions

After completing this lesson, you should be able to answer:

1. What does `pwd` do?
2. What does `ls` do?
3. Why do we use `ls` for listings?
4. What path are you in right now?
5. How do you navigate to a new folder?
6. How do you go up one level?
7. What's the Tab key for?
8. What does `echo "text" > file.txt` do?
9. How do you read a file back?
10. How do you stop a command that's running?

You should be able to answer all 10 with confidence before moving to GitBash-0.

Common Questions

Q: Is Git Bash the same as Command Prompt or PowerShell? A: No. Git Bash uses Unix/bash commands. CMD and PowerShell use Windows commands. This curriculum teaches bash (Unix) style.

Q: Why is my screen reader not reading the output? A: This is common. Use `command > file.txt` to save output to a file, then open it with Notepad for reliable reading.

Q: What if I type something wrong? A: Just press Enter and you'll see an error message. Type the correct command on the next line. No harm done!

Q: How do I get help with a command? A: Type `man command-name` (we'll cover this in GitBash-0)

Q: Why do paths look different in Git Bash? A: Git Bash uses Unix-style paths with forward slashes, not Windows backslashes. Don't worry - you'll get used to it quickly.

Path Comparison: Windows vs Git Bash

Style	Example	Where Used
Windows	`C:\Users\Name\Documents`	CMD, PowerShell, Windows Explorer
Git Bash	`/c/Users/Name/Documents`	Git Bash, macOS Terminal, Linux

In Git Bash:

- `C:\` becomes `/c/`
- Backslashes \ become forward slashes /
- Otherwise identical

Next Steps

Once you're comfortable with these basics:

- Move to GitBash-0: Getting Started for deeper path understanding
- Then continue through GitBash-1 through GitBash-5 for full terminal mastery

Resources

- Git Bash Installation Guide: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- Bash Basics: https://www.gnu.org/software/bash/manual/bash.html#Basic-Shell-Features
- NVDA Screen Reader: https://www.nvaccess.org/
- JAWS Screen Reader: https://www.freedomscientific.com/products/software/jaws/

Troubleshooting

Issue	Solution
Git Bash won't open	Make sure it's installed; search for "Git Bash" in Start menu
Can't hear the output	Try redirecting to a file: `command > output.txt`
Tab completion not working	Make sure you typed at least one character before pressing Tab

Issue	Solution
Command not found	Make sure you spelled it correctly; try `man` for available commands
Stuck in a command	Press Ctrl+C to stop it

Still stuck? The checkpoint questions and exercises are your best teacher. Work through them multiple times until comfortable.

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

GitBash-0: Getting Started - Layout, Paths, and the Shell

Estimated time: 20-30 minutes

Learning Objectives

- Launch Git Bash and locate the prompt
- Understand Unix-style path notation and shortcuts (~, `./`, `../`)
- Use tab completion to navigate quickly

Materials

- Computer with Git Bash installed
- Editor (Notepad/VS Code)

Step-by-step Tasks

1. Open Git Bash and note the prompt (it includes the current path).
2. Run `pwd` and say or note the printed path.
3. Use `ls` to list names in your home directory.
4. Practice `cd Documents`, `cd ../` and `cd ~` until comfortable.
5. Try tab-completion: type `cd ~/D` and press Tab.

Checkpoints

- Confirm you can state your current path and move to `Documents`.

Quiz - Lesson GitBash.0

1. What is a path?
2. What does ~ mean?

3. How do you autocomplete a path?
4. How do you go up one directory?
5. What command lists file names?
6. True or False: Git Bash uses backslashes (`\`) in paths like Windows CMD.
7. Explain the difference between an absolute path and a relative path.
8. If you are in `/c/Users/YourName/Documents` and you type `cd ../`, where do you end up?
9. What happens when you press Tab while typing a folder name in Git Bash?
10. Describe a practical reason why understanding paths is important for a 3D printing workflow.
11. What does `./` mean in a path, and when would you use it?
12. If a folder path contains spaces (e.g., `My Projects`), how do you navigate to it with `cd`?
13. Explain what the prompt `YourName@COMPUTER MINGW64 ~/Documents $` tells you about your current state.
14. How would you navigate to your home directory from any location using a single command?
15. What is the advantage of using relative paths (like `../`) versus absolute paths in automation scripts?

Extension Problems

1. Create a nested folder and practice `cd` into it by typing partial names and using Tab.
2. Use `ls -la` to list all files including hidden ones in a folder.
3. Save `pwd` output to a file and open it in Notepad.
4. Try `cd` into a folder whose name contains spaces; observe how quotes are handled.
5. Create a short note file and open it from Git Bash.
6. Build a folder structure that mirrors your project organization; navigate to each level and document the path.
7. Create a script that prints your current path and the total number of files in it; run it from different locations.
8. Investigate special paths (e.g., `$HOME`, `$USER`); write down what each contains and when you'd use them.
9. Compare absolute vs. relative paths by navigating to the same folder using each method; explain which is easier for automation.
10. Create a bash function that changes to a frequently-used folder and lists its contents in one command; test it from different starting locations.
11. Navigate to three different locations and at each one note the prompt, the path from `pwd`, and verify you understand what each shows.
12. Create a complex folder tree (at least 5 levels deep) and navigate it using only relative paths; verify your location at each step.
13. Document all shortcuts you know (`~`, `./`, `../`, `$HOME`) and demonstrate each one works as expected.
14. Write a guide for a peer on how to understand the Git Bash prompt and

path notation without using GUI file explorer.
15. Create a troubleshooting flowchart: if someone says ”I don't know where I am,” what commands do you give them to find out?

References

- GNU. (2024). *Bash Manual*. https://example.com
- Git SCM. (2024). *Git Bash documentation*. https://example.com
- Linux Foundation. (2024). *The Linux Command Line*. https://example.com

Helpful Resources

- Bash Manual - Navigation
- Git Bash Basics
- Linux Path Guide
- Understanding Bash Prompts
- Tab Completion in Bash

GitBash-1: Navigation - Moving Around Your File System

Duration: 1 class period **Prerequisite:** GitBash-0 (Getting Started)

Learning Objectives

By the end of this lesson, you will be able to:

- Use `pwd` to print your current location
- Use `cd` to move between directories
- Use `ls` (and its flags) to list files and folders
- Use wildcards `*` and `?` to filter listings
- Navigate relative vs. absolute paths
- Search for files by name and extension

Materials

- Git Bash
- Text editor (Notepad or VS Code)

Commands Covered in This Lesson

Command	What It Does
`pwd`	Print Working Directory - shows where you are
`cd path`	Change Directory - move to a new location

Command	What It Does
`ls`	List - shows files and folders in current location
`ls -1`	List names only, one per line (screen reader friendly)
`ls -1 -F`	List names with type indicators (/ for directories)
`ls *.extension`	List files matching a pattern

`pwd` - Where Am I?

Type `pwd` and press `Enter`. Git Bash prints the full path to your current location.

```
pwd
# Output: /c/Users/YourName
```

When to use: Always run this if you're unsure of your current location.

`cd` - Changing Directories

`cd` stands for ”change directory.”

```
# Go to Documents
cd Documents

# Go up one level to parent directory
cd ..

# Go to root of file system
cd /

# Go to home directory
cd ~

# Go to a specific path
cd /c/Users/YourName/Documents/3D_Projects

# Go to previous directory
cd -
```

`ls` - Listing Files and Folders

Use `ls -1` for screen reader compatibility — names only, one per line.

```
# List all files and folders (names only, one per line)
ls -1

# List only files (no hidden, no directories)
ls -1 -p | grep -v /

# List only directories
ls -1 -d */
```

Wildcards - Finding Files by Pattern

Wildcards help you find files without typing the full name.

*** (asterisk)** matches any number of characters:

```
# List all .scad files
ls *.scad

# List all files starting with "part"
ls part*

# List all files ending with "_final"
ls *_final*
```

? (question mark) matches exactly one character:

```
# Find files like model1.scad, model2.scad (but not model12.scad)
ls model?.scad
```

Step-by-step Practice

1. Run `pwd` and confirm your location
2. Move to `Documents`: `cd Documents`
3. Confirm you moved: `pwd`
4. List files and folders: `ls -1`
5. List only files: `ls -1 -p | grep -v /`
6. Go back up: `cd ..`
7. Search for files: `ls *.txt`

Checkpoints

After this lesson, you should be able to:

- ☐ Navigate to any folder using `cd`
- ☐ Confirm your location with `pwd`
- ☐ List files and folders with `ls -1`
- ☐ Use wildcards to find files by pattern
- ☐ Move between absolute and relative paths confidently

Quiz - Lesson GitBash.1

1. What does `pwd` show?
2. How do you list directories only with `ls`?
3. What wildcard matches any number of characters?
4. How do you list files with the `.scad` extension?
5. Give an example of an absolute path and a relative path.
6. True or False: The `*` wildcard matches exactly one character.

7. Explain the difference between `ls -1` and `ls -1 -d */`.
8. Write a command that would list all `.txt` files in your Documents folder using a wildcard.
9. How would you search for files containing "part" in their name across multiple files?
10. Describe a practical scenario where using wildcards saves time in a 3D printing workflow.
11. What happens when you use `ls model?.scad` versus `ls model*.scad`?
12. How would you navigate to a folder whose name contains both spaces and special characters?
13. If you're in `~/Documents/Projects/3D` and you want to go to `~/Documents/Resources`, what command would you use?
14. Write a command sequence that navigates to the Downloads folder, lists only files, then returns to home.
15. Explain the purpose of using `ls -1` specifically in a screen reader context.

Extension Problems

1. Write a one-line command that lists `.scad` files and saves to `scad_list.txt`.
2. Use `ls -1 ~/Documents | less` to page through long listings.
3. Combine `ls` with `grep` to search for a filename pattern.
4. Create a shortcut alias in the session for a long path and test it.
5. Practice tab-completion in a directory with many similarly named files.
6. Build a bash script that recursively lists all `.scad` and `.stl` files in a directory tree; save the results to a file.
7. Compare the output of `ls`, `ls -1`, `ls -la`, and `ls -1 -d */` to understand the flags; document what each command does.
8. Create a filtering command that displays only files modified in the last 7 days; test it on your documents folder.
9. Write a non-visual guide to Git Bash navigation; include descriptions of common patterns and how to verify directory contents audibly.
10. Develop a navigation workflow for a typical 3D printing project: move between CAD, slicing, and print-log folders efficiently; document the commands.
11. Create a complex wildcard search: find all files in a folder and subfolders that match multiple patterns (e.g., `*_v1.*` or `*_final.*`).
12. Build a script that navigates through a folder tree, counts files at each level, and reports the structure.
13. Document the output differences between `ls`, `ls -1`, `ls -la`, and `ls -1 -d */`; explain when to use each.
14. Create a navigation "cheat sheet" as a bash script that prints common paths and how to navigate to them.
15. Design a project folder structure on your computer, document each path, then create a script that validates all folders exist.

References

- GNU. (2024). *ls command reference*. https://example.com
- GNU. (2024). *Bash wildcards and globbing*. https://example.com
- GNU. (2024). *Navigation best practices in Bash*. https://example.com

Helpful Resources

- ls Command Reference
- Bash Wildcards and Globbing
- Navigation Best Practices
- Relative and Absolute Paths in Bash
- Screen Reader Tips for Git Bash

GitBash-2: File and Folder Manipulation

Estimated time: 30-45 minutes

Learning Objectives

- Create, copy, move, and delete files and folders from Git Bash
- Use `touch`, `mkdir`, `cp`, `mv`, `rm`, and `rmdir` safely
- Understand when operations are permanent and how to confirm results

Materials

- Git Bash
- Small practice folder for exercises

Step-by-step Tasks

1. Create a practice directory: `mkdir ~/Documents/GitBash_Practice` and `cd` into it.
2. Create two files: `touch file1.txt` and `touch file2.txt`.
3. Copy `file1.txt` to `file1_backup.txt` with `cp` and confirm with `ls -1`.
4. Rename `file2.txt` to `notes.txt` using `mv` and confirm.
5. Delete `file1.txt` with `rm` and verify the backup remains.

Checkpoints

- After step 3 you should see both the original and the backup file.

Quiz - Lesson GitBash.2

1. How do you create an empty file from Git Bash?
2. What command copies a file?

3. How do you rename a file?
4. What does `rm -r` do?
5. Why is `rm` potentially dangerous?
6. True or False: `cp` requires the `-r` flag to copy both files and folders.
7. Explain the difference between `rm` and `rmdir`.
8. If you delete a file with `rm`, can you recover it from Git Bash?
9. Write a command that would copy an entire folder and all its contents to a new location.
10. Describe a practical safety check you would perform before running `rm -r` on a folder.
11. What happens if you `cp` a file to a destination where a file with the same name already exists? How would you handle this safely?
12. Compare `mv old_name.txt new_name.txt` vs `mv old_name.txt ~/Documents/new_name.txt`. What is the key difference?
13. Design a workflow to safely delete 50 files matching the pattern `*.bak` from a folder containing 500 files. What commands and verifications would you use?
14. Explain how you could back up all `.scad` files from a project folder into a timestamped backup folder in one command.
15. When organizing a 3D printing project, you need to move completed designs to an archive folder and delete failed prototypes. How would you structure this as a safe, auditable process?

Extension Problems

1. Create a folder tree and copy it to a new location with `cp -r`.
2. Write a one-line command that creates three files named `a`, `b`, `c` and lists them.
3. Move a file into a new folder and confirm the move.
4. Use wildcards to delete files matching a pattern in a safe test folder.
5. Export a listing of the practice folder to `practice_listing.txt`.
6. Create a backup shell script that copies all `.scad` files from your project folder to a backup folder with timestamp naming.
7. Build a safe deletion workflow: list files matching a pattern, verify count, then delete with confirmation; document the steps.
8. Write a bash script that organizes files by extension into subfolders; test it on a sample folder tree.
9. Create a file operation audit trail: log all copy, move, and delete operations to a text file for review.
10. Develop a project template generator: a bash script that creates a standard folder structure for a new 3D printing project with essential subfolders.
11. Implement a file conflict handler: write a bash script that handles cases where `cp` would overwrite an existing file by renaming the existing file with a timestamp before copying.
12. Create a batch rename operation: use a script to rename all files in a folder

from `old_prefix_*` to `new_prefix_*`; test with actual files and verify the results.
13. Build a folder comparison tool: list all files in two folders and identify which files exist in one but not the other; output to a report.
14. Write a destructive operation validator: before executing `rm -r`, create a script that lists exactly what will be deleted, shows file counts by type, and requires explicit user confirmation to proceed.
15. Design a complete project lifecycle workflow: create folders for active projects, completed designs, and archive; include move operations between folders, backup steps, and verification that all files arrive intact.

References

- GNU. (2024). *touch command reference*. https://example.com
- GNU. (2024). *cp and mv commands*. https://example.com
- GNU. (2024). *File system operations guide*. https://example.com

Helpful Resources

- touch Command Reference
- cp Command Reference
- mv Command Reference
- rm Command Reference
- Safe Deletion Practices

GitBash-3: Input, Output, and Piping

Duration: 1 class period **Prerequisite:** GitBash-2 (File and Folder Manipulation)

Learning Objectives

By the end of this lesson, you will be able to:

- Use `echo` to print text to the screen
- Use `cat` to read file contents
- Use > to redirect output into a file
- Use | (pipe) to send one command's output to another
- Copy output to the clipboard with `clip`
- Open files with a text editor from the command line

Commands Covered

Command	What It Does
`echo "text"`	Print text to the screen
`cat filename`	Print the contents of a file
`> filename`	Redirect output into a file (overwrites)
`>> filename`	Append output to a file (adds to end)
`|`	Pipe - send output from one command to the next
`clip`	Copy piped input to the Windows clipboard (Git Bash)
`notepad.exe filename`	Open a file in Notepad

`echo` - Printing Text

`echo` prints text to the screen. It is useful for testing, for writing text into files, and for understanding how piping works.

```
echo "Hello, World"
echo "This is a test"
```

`cat` - Reading Files

`cat` prints the contents of a file to the screen.

```
# Read a text file
cat ~/Documents/notes.txt

# Read an OpenSCAD file
cat ~/Documents/OpenSCAD_Projects/project0.scad
```

With a long file, use `cat filename | less` to read it page by page (press `Space` to advance, `Q` to quit).

> - Redirecting Output to a File

The > symbol redirects output from the screen into a file instead.

```
# Create a file with a single line
echo "Author: Your Name" > header.txt

# Confirm the file was created and has content
cat header.txt
```

Warning: > overwrites the file if it already exists. Use >> to append instead:

```
echo "Date: 2025" >> header.txt
echo "Project: Floor Marker" >> header.txt
cat header.txt
```

| - Piping

The pipe symbol `|` sends the output of one command to the input of the next. This lets you chain commands together.

```
# List files and send the list to clip (copies to clipboard)
ls -1 | clip

# Now paste with Ctrl + V anywhere
```

```
# Search within a file's contents using grep
cat project0.scad | grep "cube"
```

`clip` - Copying to Clipboard

`clip` takes whatever is piped to it and puts it on the Windows clipboard.

```
# Copy your current directory path to the clipboard
pwd | clip

# Copy a file listing to clipboard
ls -1 | clip

# Copy the contents of a file to clipboard
cat notes.txt | clip
```

After any of these, press `Ctrl + V` in any application to paste.

Opening Files in Notepad

```
# Open a file in Notepad
notepad.exe ~/Documents/notes.txt

# Open a .scad file
notepad.exe ~/Documents/OpenSCAD_Projects/project0.scad

# Create a new file and open it
touch new_notes.txt
notepad.exe new_notes.txt
```

Step-by-step Tasks

1. Create `practice.txt` with three lines using `echo` and >/>>.

2. Read the file with `cat practice.txt`.
3. Pipe the file into `grep` to search for a word.
4. Copy the file contents to clipboard with `cat practice.txt | clip`.
5. Redirect `ls -1` into `list.txt` and open it in Notepad.

Checkpoints

- After step 3 you should be able to find a keyword using piping.

Quiz - Lesson GitBash.3

1. What is the difference between > and >>?
2. What does the pipe `|` do?
3. How do you copy output to the clipboard?
4. How would you page through long output?
5. How do you suppress output (send it to `/dev/null`)?
6. True or False: The pipe operator `|` connects the output of one command to the input of another.
7. Explain why redirecting output to a file is useful for screen reader users.
8. Write a command that would search for the word ”sphere” in all `.scad` files in a directory.
9. How would you count the number of lines in a file using bash piping?
10. Describe a practical scenario in 3D printing where you would pipe or redirect command output.
11. What would be the difference in output between `echo "test" > file.txt` (run twice) vs `echo "test" >> file.txt` (run twice)? Show the expected file contents.
12. Design a three-step piping chain: read a file, filter for specific content, and save the results; explain what each pipe does.
13. You have a 500-line `.scad` file and need to find all instances of `sphere()` and count them. Write the command.
14. Explain how `clip` is particularly valuable for screen reader users when working with file paths or long output strings.
15. Describe how you would use pipes and redirection to create a timestamped backup report of all `.stl` files in a 3D printing project folder.

Extension Problems

1. Use piping to count lines in a file (hint: `cat file.txt | wc -l`).
2. Save a long `ls -1` output and search it with `grep`.
3. Chain multiple pipes to filter and then save results.
4. Practice copying different command outputs to clipboard and pasting.
5. Create a small bash script that generates a report (counts of files by extension).
6. Build a data processing pipeline: read a text file, filter rows, and export results; document each step.

7. Write a script that pipes directory listing to count occurrences of each file extension; create a summary report.
8. Create a log analysis command: read a log file, filter for errors, and save matching lines to a separate error log.
9. Design a piping workflow for 3D printing file management: find `.stl` files, extract their names, and generate a report.
10. Develop a reusable piping function library: create bash functions for common filtering, sorting, and reporting patterns; test each with different inputs.
11. Build a complex filter pipeline: read a `.scad` file, extract lines containing specific geometry commands, count each type, and output a summary to both screen and file.
12. Create an interactive filtering tool: build a bash script that accepts a search term, pipes through multiple filters, and displays paginated results.
13. Develop a performance analysis tool: use piping to combine file listing, metadata extraction, and statistical reporting; export results to a dated report file.
14. Implement a comprehensive error-handling pipeline: read output, catch errors, log them separately, and generate a summary of successes vs failures.
15. Design and execute a real-world project backup workflow: use piping to verify file existence, count files by type, generate a backup manifest, and create audit logs — all in one integrated command pipeline.

References

- GNU. (2024). *Redirections in Bash*. https://example.com
- GNU. (2024). *grep command reference*. https://example.com
- GNU. (2024). *Bash pipeline concepts*. https://example.com

Helpful Resources

- Bash Redirections
- Piping and grep
- cat Command Reference
- wc for Counting
- Bash Pipeline Concepts

GitBash-4: Environment Variables, PATH, and Aliases

Estimated time: 30-45 minutes

Learning Objectives

- Read environment variables with `$VARNAME`
- Inspect and verify programs in the `PATH`

- Create temporary aliases and understand making them persistent via `.bashrc`

Materials

- Git Bash (with rights to edit `.bashrc`)

Step-by-step Tasks

1. Show your username and home path with `echo $USER` and `echo $HOME`.
2. Inspect `echo $PATH` and identify whether `openscad` or `code` would be found.
3. Run `which openscad` and note the result.
4. Create a temporary alias: `alias preview='openscad'` and test `preview myfile.scad`.
5. Open your profile (`notepad.exe ~/.bashrc`) and add the alias line to make it persistent.

Checkpoints

- After step 3 you can determine whether a program will be found by `PATH`.

Quiz - Lesson GitBash.4

1. How do you print an environment variable?
2. What is the purpose of `PATH`?
3. How do you check whether `openscad` is available?
4. How do you create a temporary alias?
5. Where would you make an alias permanent?
6. True or False: Environment variables in bash are case-sensitive.
7. Explain why having a program in your PATH is useful compared to always using its full file path.
8. Write a command that would create an alias called `slicer` for the OpenSCAD executable.
9. What file would you edit to make an alias persist across Git Bash sessions?
10. Describe a practical benefit of using the `$TMPDIR` or `/tmp` directory for temporary files in a 3D printing workflow.
11. You have a custom script at `~/scripts/backup_models.sh` that you want to run from anywhere as `backup-now`. What steps would you take to make this work?
12. Explain the difference between setting an environment variable in the current session with `export` vs. adding it to `.bashrc` for permanence.
13. Design a `.bashrc` strategy for managing multiple 3D printing projects, each with different tool paths and directories; show how to structure environment variables for each.

14. If a program is not found by `which`, what are the possible reasons, and how would you troubleshoot?
15. Describe how you would verify that your `.bashrc` is loading correctly and how to debug issues if aliases or environment variables don't appear after restarting Git Bash.

Extension Problems

1. Add a folder to PATH for a test program (describe steps; do not change system PATH without admin).
2. Create a short `.bashrc` snippet that sets two aliases and test re-opening Git Bash.
3. Use `which` to list the path for several common programs.
4. Explore `$TMPDIR` or `/tmp` and create a file there.
5. Save a copy of your current PATH to a text file and examine it in your editor.
6. Create a `.bashrc` script that loads custom aliases and environment variables for your 3D printing workflow; test it in a new session.
7. Build a "project profile" that sets environment variables for CAD, slicing, and print directories; switch between profiles for different projects.
8. Write a script that audits your current environment variables and creates a summary report of what's set and why.
9. Design a custom alias system for common 3D printing commands; document the aliases and their purposes.
10. Create a profile migration guide: document how to export and import your `.bashrc` across machines for consistent workflows.
11. Implement a safe PATH modification script: create a utility that allows you to add/remove directories from PATH for the current session only; show how to make it permanent in `.bashrc`.
12. Build a comprehensive `.bashrc` framework with modular sourcing: create separate `.sh` files for aliases, environment variables, and functions; have your main `.bashrc` source all of them.
13. Develop an environment validation tool: write a bash script that checks whether all required programs (OpenSCAD, slicers, etc.) are accessible via PATH; report findings and suggest fixes.
14. Create a project-switching alias system: design a function that changes all environment variables and aliases based on the current project; test switching between multiple projects.
15. Build a `.bashrc` troubleshooting guide: create a script that exports your current environment state (variables, aliases, PATH) to a timestamped file, allowing you to compare states before and after changes.

References

- GNU. (2024). *Environment variables in Bash*. https://example.com
- GNU. (2024). *alias command reference*. https://example.com
- GNU. (2024). *Creating and using Bash profiles*. https://example.com

Helpful Resources

- Environment Variables in Bash
- Understanding the PATH Variable
- Bash alias Reference
- Creating a Bash Profile (.bashrc)
- which Command for Locating Programs

GitBash-5: Filling in the Gaps - Shell Profiles, History, and Useful Tricks

Estimated time: 30-45 minutes

Learning Objectives

- Use history and abort commands (`history`, `Ctrl+R`, `Ctrl+C`)
- Inspect and edit your `.bashrc` profile for persistent settings
- Run programs by full path using `./` or absolute paths
- Handle common screen reader edge cases when using the terminal

Materials

- Git Bash and an editor (Notepad/VS Code)

Step-by-step Tasks

1. Run several simple commands (e.g., `pwd`, `ls -1`, `echo hi`) then run `history` to view them.
2. Use `!<n>` to re-run a previous command by its history number (replace `<n>` with a history number).
3. Practice aborting a long-running command with `Ctrl + C` (for example, `ping google.com`).
4. Open your profile: `notepad.exe ~/.bashrc`; if it doesn't exist, create it: `touch ~/.bashrc`.
5. Add a persistent alias line to your profile (example: `alias preview='openscad'`), save, and run `source ~/.bashrc` or reopen Git Bash to verify.

Checkpoints

- After step 2 you can re-run a recent command by history number.
- After step 5 your alias should persist across sessions.

Quiz - Lesson GitBash.5

1. How do you view the command history in Git Bash?
2. Which key combination aborts a running command?

3. What does `echo $BASH_VERSION` show?
4. How does `./` help run scripts and executables in the current directory?
5. What is one strategy if terminal output stops being announced by your screen reader?
6. True or False: Using `Ctrl+C` permanently deletes any files created by the command you abort.
7. Explain the difference between `history` and `Ctrl+R` (reverse history search) in Git Bash.
8. If you add an alias to `.bashrc` but it doesn't take effect after opening a new Git Bash window, what should you verify?
9. Write a command that would run a script at the path `~/scripts/openscad_runner.sh` directly.
10. Describe a practical workflow scenario where having keyboard shortcuts (aliases) in your `.bashrc` would save time.
11. Explain how to re-run the 5th command from your history using `!5`, and what would happen if that command had file operations (creates/deletes).
12. Design a `.bashrc` initialization strategy that separates utilities for different projects; explain how you would switch between them.
13. Walk through a troubleshooting workflow: your screen reader stops announcing output after running a long command. What steps would you take to diagnose and resolve the issue?
14. Create a safety checkpoint system: before any destructive operation (mass delete, overwrite), how would you use `.bashrc` functions and history to verify the command is correct?
15. Develop a comprehensive capstone scenario: integrate everything from GitBash-0 through GitBash-5 (navigation, file operations, piping, environment setup, history) to design an automated 3D printing project workflow with error handling and logging.

Extension Problems

1. Add an alias and an environment variable change to your `.bashrc` and document the behavior after reopening Git Bash.
2. Create a short bash script that automates creating a project folder and an initial `.scad` file.
3. Experiment with running OpenSCAD by full path using `./` and by placing it in PATH; compare results.
4. Practice redirecting `man ls` output to a file and reading it in Notepad for screen reader clarity.
5. Document three screen reader troubleshooting steps you used and when they helped.
6. Build a comprehensive `.bashrc` that includes aliases, environment variables, and helper functions for your 3D printing workflow.
7. Create a bash script that troubleshoots common Git Bash issues (module loading, permission errors, command not found); test at least three scenarios.

8. Write a bash function that coordinates multiple tasks: creates a project folder, starts OpenSCAD, and opens a notes file.
9. Design a screen-reader accessibility guide for Git Bash: document commands, outputs, and accessible navigation patterns.
10. Develop an advanced Git Bash workflow: implement error handling, logging, and confirmation prompts for risky operations.
11. Implement an ”undo” system using history: create a function that logs destructive commands (`rm`, `mv`, `cp`) and allows you to review the last operation.
12. Build a `.bashrc` debugger: create a script that compares two Git Bash sessions' environment states (variables, aliases, functions) to identify what loaded/failed to load.
13. Develop a multi-project profile manager: design a system where you can switch entire environments (paths, aliases, variables) for different 3D printing projects by running a single command.
14. Create a comprehensive accessibility analyzer: write a bash script that tests whether key Git Bash commands produce screen-reader-friendly output; document workarounds for commands that don't.
15. Design a complete capstone project: build an integrated automation suite that manages a 3D printing workflow (project setup, file organization, CAD/slicing tool automation, output logging, error recovery, and audit trails) with full error handling and documentation.

References

- GNU. (2024). *Bash history and recall functionality*. https://example.com
- GNU. (2024). *Understanding and creating Bash profiles*. https://example.com
- GNU. (2024). *Running scripts and executables in Bash*. https://example.com

Helpful Resources

- Bash History and Recall
- Understanding .bashrc
- history Command Reference
- Running Scripts with ./
- Screen Reader Tips and Tricks

GitBash-6: Advanced Terminal Techniques - Shell Scripts, Functions & Professional Workflows

Duration: 4-4.5 hours (for screen reader users) **Prerequisites:** Complete GitBash-0 through GitBash-5 **Skill Level:** Advanced intermediate

This lesson extends Git Bash skills to professional-level workflows. You'll learn to automate complex tasks, write reusable shell scripts, and integrate tools for 3D printing workflows.

Learning Objectives

By the end of this lesson, you will be able to:

- Create and run shell scripts (.sh files)
- Write functions that accept parameters
- Use loops to repeat tasks automatically
- Automate batch processing of 3D models
- Debug scripts when something goes wrong
- Create professional workflows combining multiple tools

Shell Script Basics

What's a Shell Script? A shell script (`.sh`) contains multiple bash commands that run in sequence. Instead of typing commands one by one, you put them in a file and run them all at once.

Why use shell scripts?

- Repeatability: Run the same task 100 times identically
- Documentation: Commands are written down for reference
- Complexity: Combine many commands logically
- Automation: Schedule scripts to run automatically

Creating Your First Shell Script

Step 1: Open a text editor

```
notepad.exe my-first-script.sh
```

Step 2: Type this script

```
#!/bin/bash
# This is a comment - screen readers will read it
echo "Script is running!"
pwd
ls -1
echo "Script is done!"
```

Step 3: Save the file

- In Notepad: Ctrl+S
- Make sure filename ends in `.sh`
- Save in an easy-to-find location (like Documents)

Step 4: Make it executable and run the script

```bash
chmod +x my-first-script.sh
./my-first-script.sh
```

What happens: Bash runs each command in sequence and shows output.

Important: The Shebang Line The `#!/bin/bash` at the top of your script (called a "shebang") tells the system which shell to use to run the script.

```bash
#!/bin/bash
# Now bash runs every command below
echo "Hello!"
```

Variables and Parameters

Using Variables Variables store values you want to use later.

Example script:

```bash
#!/bin/bash
mypath="$HOME/Documents"
cd "$mypath"
echo "I am now in:"
pwd
ls -1
```

Breaking it down:

- `mypath="..."` assigns the variable (no spaces around =)
- `"$mypath"` uses the variable (quote it to handle spaces in paths)
- Variables in bash are always referenced with $

Functions with Parameters A function is reusable code that you can run with different inputs.

Example: A function that lists files in a folder

```bash
#!/bin/bash
list_folder() {
    local path="$1"
    echo "Contents of: $path"
    cd "$path"
    ls -1
}

# Use the function:
list_folder "$HOME/Documents"
list_folder "$HOME/Downloads"
```

What's happening:

- `list_folder()` defines the function
- `local path="$1"` assigns the first argument to a local variable
- $1 is the first argument passed to the function

- Call the function with `list_folder "path/to/folder"`

Screen reader tip: When you call a function, bash will announce the results just like any command.

Loops - Repeating Tasks

Loop Over Files Imagine you have 10 SCAD files and want to print their contents. You could do it 10 times manually, or use a loop.

Example: Print every .scad file in a folder

```bash
#!/bin/bash
for file in *.scad; do
   echo "=== File: $file ==="
   cat "$file"
   echo ""
done
```

What's happening:

- `for file in *.scad; do` iterates over each `.scad` file
- `$file` holds the current filename
- `done` ends the loop
- Inside the loop, do something with each `$file`

Loop with a Counter **Example: Do something 5 times**

```bash
#!/bin/bash
for i in $(seq 1 5); do
   echo "This is iteration number $i"
   # Do something here
done
```

What's happening:

- `for i in $(seq 1 5)` loops with `i` from 1 to 5
- `$i` is the counter variable

Real-World Example - Batch Processing SCAD Files

Scenario You have 10 OpenSCAD (.scad) files in a folder. You want to:

1. List them all
2. Check how many there are
3. For each one, verify it exists

The Script

```bash
#!/bin/bash
scad_folder="$HOME/Documents/3D_Projects"
count=0

echo "Processing SCAD files in: $scad_folder"
echo ""

for file in "$scad_folder"/*.scad; do
    if [ -f "$file" ]; then
        echo " Found: $(basename "$file")"
        count=$((count + 1))
    else
        echo " Missing: $(basename "$file")"
    fi
done

echo ""
echo "Total files found: $count"
echo "Batch processing complete!"
```

Breaking it down:

- `scad_folder="..."` = where to look
- `for file in "$scad_folder"/*.scad` = find all .scad files
- `if [ -f "$file" ]` = check if file exists and is a regular file
- `basename "$file"` = just the filename (not the full path)
- `count=$((count + 1))` = increment the counter

Running the Script

1. Save as `batch-process.sh`
2. Edit `scad_folder` to match your real folder
3. Make it executable and run it:

```
chmod +x batch-process.sh
./batch-process.sh
```

Screen reader output:

```
Processing SCAD files in: /c/Users/YourName/Documents/3D_Projects

  Found: model1.scad
  Found: model2.scad
  Found: model3.scad
[... more files ...]

Total files found: 10
Batch processing complete!
```

Error Handling

Try-Style Checks with Exit Codes What if something goes wrong? Use exit code checks:

Example:

```bash
#!/bin/bash
file="$HOME/nonexistent/path/file.txt"

if cat "$file" 2>/dev/null; then
    echo "File read successfully"
else
    echo "Error: Could not read file"
    echo "File path was: $file"
fi
```

What's happening:

- `2>/dev/null` suppresses error messages from `cat`
- `if cat ...; then` checks whether the command succeeded
- `else` handles the failure gracefully

Screen reader advantage: Errors are announced clearly instead of crashing silently.

Validating Input **Example: Make sure a folder exists before processing**

```bash
#!/bin/bash
process_folder() {
    local folder_path="$1"

    if [ ! -d "$folder_path" ]; then
        echo "Error: Folder does not exist: $folder_path"
        return 1
    fi

    echo "Processing folder: $folder_path"
    ls -1 "$folder_path"
}

process_folder "$HOME/Documents"
```

What's happening:

- `[ ! -d "$folder_path" ]` checks if the path is NOT a directory
- `return 1` exits the function early with a non-zero (error) status

---

Debugging Shell Scripts

Common Errors and Solutions

Error 1: "Command not found" **Cause:** Typo in command name

Fix: Check spelling

```bash
# Wrong:
ech "hello"

# Correct:
echo "hello"
```

Error 2: "Variable is empty" **Cause:** Variable was never assigned or has a typo

Fix: Make sure variable is set before using it

```
myvar="hello" # Set first
echo "$myvar" # Then use
```

Error 3: "No such file or directory" **Cause:** Wrong folder path

Fix: Verify path exists

```
# Check if path exists:
if [ -d "$HOME/Documents" ]; then
    echo "Path exists"
fi
```

Error 4: "Permission denied" **Cause:** Script not executable, or no write permission

Fix: Make the script executable, or check file permissions

```
chmod +x my-script.sh
```

Debugging Technique: Trace Output with `set -x` Add `set -x` at the top of your script to print each command before it runs:

```
#!/bin/bash
set -x # Enable trace mode
path_var="$HOME/Documents"
echo "Starting script. Path is: $path_var"

for file in "$path_var"/*; do
    echo "Processing: $file"
    # Do something with $file
    echo "Done with: $file"
done

echo "Script complete"
```

Your screen reader will announce each step, so you know where errors happen.

Creating Professional Workflows

Example 1: Automated Project Setup **Scenario:** You start a new 3D printing project regularly. Instead of creating folders manually:

```
#!/bin/bash
read -p "Enter project name: " project_name
base_folder="$HOME/Documents/3D_Projects"
project_folder="$base_folder/$project_name"

# Create folder structure
mkdir -p "$project_folder"
mkdir -p "$project_folder/designs"
mkdir -p "$project_folder/output"
mkdir -p "$project_folder/notes"

# Create a README
cat > "$project_folder/README.txt" << EOF
# $project_name

Created: $(date)

## Designs
All .scad files go here.

## Output
STL and other exports go here.

## Notes
Project notes and observations.
EOF

echo "Project setup complete: $project_folder"
```

What it does:

- Prompts for a project name
- Creates folder structure for a new project
- Sets up subfolders for designs, output, notes
- Creates a README file automatically

Example 2: Batch File Verification **Scenario:** Before processing, verify all required files exist:

```
#!/bin/bash
verify_project() {
    local project_folder="$1"
    local required_items=("README.txt" "designs" "output" "notes")
    local all_good=true

    for item in "${required_items[@]}"; do
        local path="$project_folder/$item"
        if [ -e "$path" ]; then
            echo " Found: $item"
        else
            echo " Missing: $item"
            all_good=false
        fi
    done

    if $all_good; then
        echo "All checks passed!"
        return 0
    else
        echo "Some files are missing!"
        return 1
    fi
}

# Use it:
project="$HOME/Documents/3D_Projects/MyKeychain"
if verify_project "$project"; then
    echo "Safe to proceed with processing"
fi
```

Screen Reader Tips for Shell Scripts

Making Script Output Readable **Problem:** Script runs but output scrolls too fast or is hard to follow

Solution 1: Save to file

```
./my-script.sh > output.txt
notepad.exe output.txt
```

Solution 2: Use echo with clear sections

```
echo "========== STARTING =========="
echo ""
# ... script ...
echo ""
echo "========== COMPLETE =========="
```

Solution 3: Pause between major sections

```
echo "Pausing... Press Enter to continue"
read
```

Your screen reader will announce the pause, give you time to read output.

Announcing Progress **For long-running scripts:**

```
#!/bin/bash
count=0
total=$(ls *.scad | wc -l)

for file in *.scad; do
    count=$((count + 1))
    echo "Processing $count of $total: $file"
    # Do something with $file
done

echo "All $count files processed!"
```

Practice Exercises

Exercise 1: Your First Shell Script **Goal:** Create and run a simple shell script

Steps:

1. Create file: `notepad.exe hello.sh`
2. Type:

```
#!/bin/bash
echo "Hello from my first Git Bash shell script!"
pwd
ls -1
```

3. Save, make executable, and run:

```
chmod +x hello.sh
```

```
./hello.sh
```

Checkpoint: You should see output for each command.

Exercise 2: Script with a Variable **Goal:** Use a variable to make the script flexible

Steps:

1. Create file: `notepad.exe smart-listing.sh`
2. Type:

```bash
#!/bin/bash
target_folder="$HOME/Documents"
echo "Listing contents of: $target_folder"
ls -1 "$target_folder"
```

3. Edit `target_folder` to a real folder on your computer
4. Run:

```
chmod +x smart-listing.sh
./smart-listing.sh
```

Checkpoint: You should see listing of that specific folder.

Exercise 3: Function **Goal:** Create a reusable function

Steps:

1. Create file: `notepad.exe navigate.sh`
2. Type:

```bash
#!/bin/bash
go_to() {
    local path="$1"
    if [ -d "$path" ]; then
        cd "$path"
        echo "Now in: $(pwd)"
        echo "Contents:"
        ls -1
    else
        echo "Path does not exist: $path"
    fi
}

# Test the function:
go_to "$HOME/Documents"
go_to "$HOME/Downloads"
```

3. Run:

```
chmod +x navigate.sh
./navigate.sh
```

Checkpoint: Both function calls should work, showing contents of each folder.

Exercise 4: Loop **Goal:** Use a loop to repeat an action

Steps:

1. Create file: `notepad.exe repeat.sh`
2. Type:

```bash
#!/bin/bash
echo "Demonstrating a loop:"

for i in $(seq 1 5); do
    echo "Iteration $i: Hello!"
done

echo "Loop complete!"
```

3. Run:

```bash
chmod +x repeat.sh
./repeat.sh
```

Checkpoint: Should print ”Iteration 1” through ”Iteration 5”.

Exercise 5: Real-World Script **Goal:** Create a useful script for a real task

Steps:

1. Create a folder: `mkdir ~/Documents/TestFiles`
2. Create some test files:

```bash
echo "test" > ~/Documents/TestFiles/file1.txt
echo "test" > ~/Documents/TestFiles/file2.txt
echo "test" > ~/Documents/TestFiles/file3.txt
```

3. Create script: `notepad.exe report.sh`
4. Type:

```bash
#!/bin/bash
folder="$HOME/Documents/TestFiles"
count=0

echo "=== FILE REPORT ==="
echo "Folder: $folder"
echo ""
echo "Files:"
for file in "$folder"/*; do
    echo " - $(basename "$file")"
    count=$((count + 1))
done
echo ""
echo "Total: $count files"
echo "=== END REPORT ==="
```

5. Run:

```bash
chmod +x report.sh
./report.sh
```

Checkpoint: Should show report of all files in the test folder.

Quiz - Lesson GitBash.6

1. What is a shell script?
2. What file extension do bash shell scripts use?
3. What is a variable in bash and how do you create one?
4. What is a function and why would you use one?
5. How do you run a shell script?
6. What is a `for` loop and what does `for file in *.scad; do` do?
7. What does `[ -f "$file" ]` check?
8. How do you handle errors in a bash script?
9. When would you use `if [ ! -d "$path" ]; then`?
10. What technique makes shell script output readable for screen readers?

Extension Problems

1. **Auto-Backup Script:** Create a bash script that copies all files from one folder to another, announcing progress
2. **File Counter:** Write a function that counts files by extension (.txt, .scad, .stl, etc.)
3. **Folder Cleaner:** Script that deletes files older than 30 days (with user confirmation)
4. **Project Template:** Function that creates a complete project folder structure with all needed files
5. **Batch Rename:** Script that renames all files in a folder according to a pattern
6. **Log Generator:** Create a script that records what it does to a log file for later review
7. **Scheduled Task:** Set up a script to run automatically using cron or Task Scheduler
8. **File Verifier:** Check that all SCAD files in a folder have corresponding STL exports
9. **Report Generator:** Create a summary report of all projects in a folder
10. **Error Tracker:** Script that lists all commands that had errors and logs them with timestamps

---

Important Notes

- **Always test scripts on small sets of files first** before running them on important data
- **Save your work regularly** — use version naming if possible
- **Test error handling** — make sure errors don't crash silently
- **Document your scripts** — use # comments so you remember what each part does

- **Backup before batch operations** — if something goes wrong, you have the original

---

References

- **GNU Bash Scripting Guide:** https://example.com
- **Function Documentation:** https://example.com
- **Error Handling:** https://example.com
- **Loops:** https://example.com

GitBash Unit Test - Comprehensive Assessment

Estimated time: 60-90 minutes

Key Learning Outcomes Assessed

By completing this unit test, you will demonstrate:

1. Understanding of file system navigation and path concepts
2. Proficiency with file and folder manipulation commands
3. Ability to redirect and pipe command output
4. Knowledge of environment variables and aliases
5. Screen-reader accessibility best practices in terminal environments
6. Problem-solving and command chaining skills

Target Audience

Users who have completed GitBash-0 through GitBash-6 and need to demonstrate mastery of Git Bash fundamentals.

Instructions

Complete all sections below. For multiple choice, select the best answer. For short answers, write one to two sentences. For hands-on tasks, capture evidence (screenshots or output files) and submit alongside your answers.

Part A: Multiple Choice Questions (20 questions)

Select the best answer for each question. Each question is worth 1 point.

1. What is the primary purpose of the `PATH` environment variable?
    - A) Store your home directory location
    - B) Tell the shell where to find executable programs

- C) Configure the visual appearance of the terminal
- D) Store the current working directory name

2. Which command prints your current working directory in Git Bash?

   - A) `ls -1`
   - B) `cd`
   - C) `pwd`
   - D) `whoami`

3. What does the ~ symbol represent in Git Bash paths?

   - A) The root directory
   - B) The current directory
   - C) The parent directory
   - D) The home directory

4. How do you list only file names (not full details) in a way that is screen-reader friendly?

   - A) `ls`
   - B) `ls -1`
   - C) `ls -l`
   - D) `cat -1`

5. Which command creates a new empty file in Git Bash?

   - A) `mkdir filename`
   - B) `touch filename`
   - C) `new filename`
   - D) `echo filename`

6. What is the difference between > and >>?

   - A) > redirects to file, >> displays on screen
   - B) > overwrites a file, >> appends to a file
   - C) They do the same thing
   - D) > is for text, >> is for binary

7. What does the pipe operator `|` do?

   - A) Creates a folder
   - B) Sends the output of one command to the input of another
   - C) Deletes files matching a pattern
   - D) Lists all processes

8. Which command copies a file in Git Bash?

   - A) `mv`
   - B) `rm`
   - C) `cp`
   - D) `cd`

9. How do you rename a file from `oldname.txt` to `newname.txt` in Git Bash?
   - A) `cp oldname.txt newname.txt`
   - B) `mv oldname.txt newname.txt`
   - C) `rename oldname.txt newname.txt`
   - D) `rn oldname.txt newname.txt`
10. What is the purpose of `grep` in Git Bash piping?
    - A) Find files in a directory
    - B) Search for text patterns within output or files
    - C) Select a string to copy to clipboard
    - D) Select which shell to use
11. Which key allows you to autocomplete a path in Git Bash?
    - A) `Ctrl + A`
    - B) `Ctrl + E`
    - C) `Tab`
    - D) `Space`
12. How do you copy text to the Windows clipboard from Git Bash?
    - A) `cat filename > clipboard`
    - B) `cat filename | clip`
    - C) `copy filename`
    - D) `cat filename | paste`
13. What does `which openscad` do?
    - A) Opens the OpenSCAD application
    - B) Gets help about the openscad command
    - C) Locates the full path of the openscad executable
    - D) Lists all available commands
14. Which wildcard matches any single character in Git Bash?
    - A) `*`
    - B) `?`
    - C) `%`
    - D) `#`
15. What is the purpose of `./` before a script name?
    - A) Run a script in the current directory
    - B) Execute all commands in parallel
    - C) Combine multiple commands
    - D) Create an alias
16. How do you create a temporary alias in Git Bash?
    - A) `set-alias preview='openscad'`
    - B) `alias preview='openscad'`

- C) `new-alias preview openscad`
- D) `preview = openscad`

17. Where is a Git Bash alias typically stored to persist across sessions?

    - A) `C:\Program Files\Git\profile.sh`
    - B) In the `~/.bashrc` file
    - C) `~/bash_profile`
    - D) Aliases cannot be made permanent

18. How do you abort a long-running command in Git Bash?

    - A) Press `Escape`
    - B) Press `Ctrl + X`
    - C) Press `Ctrl + C`
    - D) Press `Alt + F4`

19. What command shows the history of previously run commands in Git Bash?

    - A) `history`
    - B) `get-history`
    - C) `show-history`
    - D) `bash-history`

20. How do you reload your `.bashrc` without restarting Git Bash?

    - A) Use `reload ~/.bashrc` in the terminal
    - B) Use `source ~/.bashrc`
    - C) Use the Windows Control Panel
    - D) `.bashrc` reloads automatically

Part B: Short Answer Questions (10 questions)

Answer each question in one to two sentences. Each question is worth 2 points.

1. Explain the difference between absolute and relative paths. Give one example of each.

2. Why is `ls -1` preferred over `ls` for screen reader users? Describe what flag combination you would use to list only directories.

3. What is the purpose of redirecting output to a file, and give an example of when you would use > instead of >>?

4. Describe what would happen if you ran `rm -r ~/Documents/my_folder` and why this command should be used carefully.

5. How would you search for all files with a `.scad` extension in your current directory? Write the command.

6. Explain what happens when you pipe the output of `ls -1` into `clip`. What would you do next?
7. What is an environment variable, and give one example of how you might use it in Git Bash.
8. If a program is not in your `PATH`, what two methods could you use to run it from Git Bash?
9. Describe how you would open a file in Notepad and also add a line to it from Git Bash.
10. What is one strategy you would use if your screen reader stops announcing terminal output while using Git Bash?

Part C: Hands-On Tasks (10 tasks)

Complete each task and capture evidence (screenshots, output files, or command transcripts). Each task is worth 3 points.

Tasks 1-5: File System and Navigation

1. Create a folder structure `~/Documents/GitBash_Assessment/Projects` using a single command. Capture the `ls -1` output showing the creation.
2. Create five files named `project_1.scad`, `project_2.scad`, `project_3.txt`, `notes_1.txt`, and `notes_2.txt` inside the `Projects` folder. Use wildcards to list only `.scad` files, then capture the output.
3. Copy the entire `Projects` folder to `Projects_Backup` using `cp -r`. Capture the `ls -1` output showing both folders exist.
4. Move (rename) `project_1.scad` to `project_1_final.scad`. Capture the `ls -1` output showing the renamed file.
5. Delete `notes_1.txt` and `notes_2.txt` using a single `rm` command with wildcards. Capture the final `ls -1` output.

Tasks 6-10: Advanced Operations and Scripting

6. Create a file called `my_data.txt` with at least four lines using `echo` and `>>`. Then read it with `cat my_data.txt` and capture the output.
7. Use `grep` to search for a keyword (e.g., ”project”) in `my_data.txt` and pipe the results to `clip`. Paste the results into Notepad and capture a screenshot.
8. List all files in the `Projects` folder and redirect the output to `projects_list.txt`. Open it in Notepad and capture a screenshot of the file.

9. Create a temporary alias called `myls` that runs `ls -1`, test it, and capture the output. Then explain what would be required to make it persistent.
10. Run `man ls` (or `ls --help`) and redirect the output to `help_output.txt`. Open the file in Notepad and capture a screenshot showing at least the first page of help content.

Grading Rubric

Section	Questions	Points Each	Total
Multiple Choice	20	1	20
Short Answer	10	2	20
Hands-On Tasks	10	3	30
Total	**40**	-	**70**

Passing Score: 49 points (70%)

Helpful Resources for Review

- Git Bash Command Reference
- Navigation and File System
- Using Pipes and Filtering
- Bash Profile and Aliases
- Screen Reader Accessibility Tips

Submission Checklist

☐ All 20 multiple choice questions answered
☐ All 10 short answer questions answered (1-2 sentences each)
☐ All 10 hands-on tasks completed with evidence captured
☐ Files/screenshots organized and labeled clearly
☐ Submission includes this checklist

3dMake-Accessible Design with OpenSCAD

This section covers the complete 3D design and printing workflow using OpenSCAD, 3DMake, and accessible tools. Students progress from basic primitives through advanced parametric design and stakeholder-centric projects.

Time commitment: 30-40 hours instruction + projects

Skills gained: 3D modeling, parametric design, automated workflows, tolerance testing, quality assurance

Instructional Framework for 3DMake and OpenSCAD in Secondary STEM Education

The transition from traditional, direct-manipulation computer-aided design (CAD) to programmatic, declarative modeling represents a fundamental shift in how physical objects are conceptualized and engineered. For the high school junior, this transition is not merely a change in software but a pivot toward computational thinking, where geometry is derived from mathematical logic rather than visual approximation. The 3DMake ecosystem, an open-source command-line toolchain, serves as the critical connective tissue in this lifecycle, automating the translation of source code into physical matter.[1] By integrating OpenSCADs script-based modeling with 3DMakes automation of rendering, slicing, and verification, students engage with a workflow that mirrors professional DevOps and industrial manufacturing pipelines.[2] This report provides an exhaustive pedagogical guide, safety background, and technical analysis of this ecosystem, tailored for an advanced secondary education environment.

[1]Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[2]Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

The Architecture of Programmatic Design: Introduction to 3DMake

Programmatic CAD differs from standard industry tools like Fusion 360 or SolidWorks by utilizing a "code-as-model" philosophy. OpenSCAD, the primary engine supported by 3DMake, uses a functional programming language to define three-dimensional volumes.[3] This approach is particularly robust for parametric design, where the dimensions of an object are defined as variables, allowing for instantaneous reconfiguration without manual rebuilding.[4] 3DMake enhances this by providing a unified command-line interface (CLI) to manage the entire process, from editing source files to triggering remote print jobs through interfaces like OctoPrint or Bambu Connect.[5]

The educational value of this toolchain lies in its transparency. Students are not hidden behind a proprietary user interface; they interact directly with the file system, configuration files, and API integrations. This exposure fosters a deeper understanding of how modern software interacts with hardware, a skill set increasingly vital in robotics and aerospace engineering.[6] However, the shift to a terminal-based environment requires a structured instructional approach to overcome initial barriers to entry.

Background on AI Integration and Model Descriptive Feedback

The integration of artificial intelligence into the 3DMake workflow represents a significant advancement in democratizing CAD for students with varying levels of spatial reasoning skills. The 3dm info command acts as a multimodal bridge between the deterministic world of OpenSCAD and the probabilistic world of LLMs.[7]

When a student executes 3dm info, the system initiates a rendering pipeline. It generates multiple viewpoints of the current model, which are then packaged as image data.[8] If the Gemini API is configured, these images are sent to the model along with a prompt that requires the AI to synthesize a 3D understanding from 2D representations. The AI's response is then returned to the terminal, providing a descriptive summary that can include the model's intended function,

[3] Programming with OpenSCAD, accessed February 18, 2026, https://programmingwithopenscad.github.io/

[4] OpenSCAD Review - Worth learning? - CadHub, accessed February 18, 2026, https://learn.cadhub.xyz/blog/openscad-review/

[5] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[6] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad

[7] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[8] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

aesthetic qualities, and potential engineering failures.[9]

The Limits of AI Spatial Reasoning

While this feature is powerful, there is a fundamental disconnect in how current LLMs process 3D data. Most models are trained on 2D images and text; they do not possess a true ”3D world model”.[10] This leads to several common failures:

- Detached Geometry: The AI might describe a ”table” even if the legs are hovering below the tabletop-a common error in OpenSCAD translation.[11]
- Scale Misinterpretation: Without a reference object in the render, the AI may misjudge the scale of the model, leading to inappropriate feedback on wall thickness.[12]
- Hallucination of Detail: The AI may describe features (like ”engraved text”) that it expects to see based on the prompt, even if the students code failed to render them.[13]

For the high school student, the takeaway is that AI is a verification assistant, not a source of truth. The deterministic rendering of OpenSCAD remains the final authority on the model's geometry. The AI is most useful as a ”sanity check” to catch obvious mistakes before wasting filament on a failed print.[14]

Occupational Health and Safety in the 3D Printing Environment

Safety in the 3DMake workflow is not limited to the digital realm. The physical act of printing involves thermal, chemical, and mechanical risks that must be addressed through rigorous institutional policies. The National Institute for Occupational Safety and Health (NIOSH) and various university environmental health departments provide a clear framework for these risks.[15]

[9] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[10] Build Great AI: LLM-Powered 3D Model Generation for 3D Printing - ZenML LLMOps Database, accessed February 18, 2026, https://www.zenml.io/llmops-database/llm-powered-3d-model-generation-for-3d-printing

[11] Build Great AI: LLM-Powered 3D Model Generation for 3D Printing - ZenML LLMOps Database, accessed February 18, 2026, https://www.zenml.io/llmops-database/llm-powered-3d-model-generation-for-3d-printing

[12] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[13] Build Great AI: LLM-Powered 3D Model Generation for 3D Printing - ZenML LLMOps Database, accessed February 18, 2026, https://www.zenml.io/llmops-database/llm-powered-3d-model-generation-for-3d-printing

[14] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[15] 3D Printer Safety - Environmental Health and Safety - The Ohio State University, accessed February 18, 2026, https://ehs.osu.edu/kb/3d-printer-safety

Chemical and Particulate Emissions

The melting of plastic filament is a thermal degradation process. ABS (Acrylonitrile Butadiene Styrene) is particularly hazardous, releasing styrene, a known respiratory irritant and potential carcinogen.[16] Even PLA (Polylactic Acid), often marketed as ”safe” and ”biodegradable,” emits millions of ultrafine particles (UFPs) per minute during extrusion.[17] These particles, smaller than 100 nanometers, can penetrate deep into the lungs and cross into the bloodstream.12

Emission Component	Primary Source Filaments	Mitigation Strategy
Styrene	ABS, ASA	Enclosed printer with carbon filtration.[18]
Formaldehyde	POM, Nylon	High-efficiency external ventilation.[19]
Ultrafine Particles	All filaments	HEPA filtration and ”20-minute” settling period.[20]
Volatile Organic Compounds	All filaments	Minimum 6 air changes per hour in the room.[21]

Physical and Mechanical Hazards

3D printers utilize high-torque stepper motors and heated elements. The extruder nozzle can reach $260^\circ\text{C}$, and the heated build plate can reach $110^\circ\text{C}$.[22] Mechanical hazards include ”pinch points” in the gantry system where fingers or loose clothing can be trapped.[23] Furthermore, post-processing activities-such as removing supports or sanding parts-introduce the risk of cuts from sharp tools and the inhalation of plastic

[16] 3D Printer Safety - Environmental Health and Safety - The Ohio State University, accessed February 18, 2026, https://ehs.osu.edu/kb/3d-printer-safety

[17] 3D Printer Safety - Environmental Health and Safety - The Ohio State University, accessed February 18, 2026, https://ehs.osu.edu/kb/3d-printer-safety

[18] 3D Printer Safety - Environmental Health and Safety - The Ohio State University, accessed February 18, 2026, https://ehs.osu.edu/kb/3d-printer-safety

[19] 3D Printer Safety - Environmental Health and Safety - The Ohio State University, accessed February 18, 2026, https://ehs.osu.edu/kb/3d-printer-safety

[20] Safe 3D Printing is for Everyone, Everywhere | NIOSH Blogs - CDC, accessed February 18, 2026, https://www.cdc.gov/niosh/blogs/2024/safe-3d-printing.html

[21] 3D Printers | Washington State Department of Health, accessed February 18, 2026, https://doh.wa.gov/community-and-environment/schools/3d-printers

[22] 3-D Printer Safety | Environmental Health & Safety | RIT, accessed February 18, 2026, https://www.rit.edu/ehs/3-d-printer-safety

[23] Safe 3D Printing is for Everyone, Everywhere | NIOSH Blogs - CDC, accessed February 18, 2026, https://www.cdc.gov/niosh/blogs/2024/safe-3d-printing.html

dust.[24]
Standard operating procedures (SOPs) for a student lab must include:

- Pre-use Inspection: Checking for frayed wires, loose belts, and a clear build surface.[25]
- Environmental Controls: Prohibiting eating or drinking in the print area to avoid ingestion of contaminants.[26]
- Emergency Response: Locating the nearest Class D fire extinguisher (for metal prints) or standard ABC extinguisher.[27]
- Post-Print Cooling: Ensuring the printer has cooled to below $30^\circ\text{C}$ before attempting to remove the model.15

Challenges Inherent in the OpenSCAD Language

OpenSCAD is often described as "the programmer's CAD," but its declarative nature and unique rendering kernel present specific hurdles for high school students. Unlike imperative languages (where you tell the computer *how* to do something), OpenSCAD is declarative.[28] This can be counterintuitive for students familiar with Python or JavaScript.

The Problem of Immutable State

In OpenSCAD, variables are not truly "variable" in the traditional sense; they are more like constants within a specific scope. If a student writes `x = 5; x = 10;`, OpenSCAD will use the last value assigned to `x` for the entire script (in this case, `x = 10`).[29] This behavior is similar to SQL or XSLT, and requires the student to adopt a functional programming mindset where geometry is defined by its state rather than its sequence of movements.[30]

[24] 3D Printer Safety - Environmental Health and Safety - The Ohio State University, accessed February 18, 2026, https://ehs.osu.edu/kb/3d-printer-safety

[25] 3D Printers | Washington State Department of Health, accessed February 18, 2026, https://doh.wa.gov/community-and-environment/schools/3d-printers

[26] 3-D Printer Safety | Environmental Health & Safety | RIT, accessed February 18, 2026, https://www.rit.edu/ehs/3-d-printer-safety

[27] 3D Printer Safety - Environmental Health & Safety - University of Tennessee, Knoxville, accessed February 18, 2026, https://ehs.utk.edu/index.php/table-of-policies-plans-procedures-guides/3d-printer-safety/

[28] Understanding the Challenges of OpenSCAD Users for 3D Printing - Thomas Pietrzak, accessed February 18, 2026, https://thomaspietrzak.com/bibliography/gonzalez24.pdf

[29] Why is there so little content and community around a tool as powerful and interesting as OpenSCAD? (beyond the awesome folks in this channel) - Reddit, accessed February 18, 2026, https://www.reddit.com/r/openscad/comments/1fxj8xv/why_is_there_so_little_content_and_community/

[30] Why is there so little content and community around a tool as powerful and interesting as OpenSCAD? (beyond the awesome folks in this channel) - Reddit, accessed February 18, 2026, https://www.reddit.com/r/openscad/comments/1fxj8xv/why_is_there_so_little_content_and_community/

Performance Bottlenecks and Functional Limits

OpenSCAD uses the CGAL (Computational Geometry Algorithms Library) as its geometry kernel. While highly accurate, CGAL is notoriously slow for certain operations.

- Minkowski Sums: This operation, often used to round corners, is computationally explosive. A simple rounded cube can take minutes to render if the resolution (defined by $fn) is too high.[^21]
- Hull Operations: The hull() function creates a convex "shrink-wrap" around objects. While faster than minkowski, it cannot be used inside certain loops and can fail if the child objects are non-manifold.[31]
- Lack of Native Filleting: Unlike modern CAD tools, OpenSCAD has no native "fillet" command. Students must manually construct these features using boolean subtractions or libraries like BOSL2.[32]

The "Absolute Coordinate" Barrier

In tools like Fusion 360, parts are often "jointed" relative to one another. OpenSCAD has no concept of relative constraints.[33] Every object is positioned in absolute $(\text{X}, \text{Y}, \text{Z})$ space. If a student moves one part, they must manually calculate and update the translation of every related part.[34] This necessitates the heavy use of variables and mathematical offsets, a process that is highly prone to human error.[35]

Technical Limitations and Risks of the 3DMake Tool

While 3DMake provides a powerful automation layer, it introduces its own set of limitations and risks.

Dependency Management and Setup Complexity

3DMake is a CLI wrapper, meaning its functionality is entirely dependent on the host machine's configuration. The ./3dm setup process is a critical point of failure.[36] If the student provides an incorrect path to the OpenSCAD executable

[31]The great thing about OpenSCAD is that it makes it easy to 3D model things which... | Hacker News, accessed February 18, 2026, https://news.ycombinator.com/item?id=46338565

[32]OpenSCAD Review - Worth learning? - CadHub, accessed February 18, 2026, https://learn.cadhub.xyz/blog/openscad-review/

[33]The great thing about OpenSCAD is that it makes it easy to 3D model things which... | Hacker News, accessed February 18, 2026, https://news.ycombinator.com/item?id=46338565

[34]The great thing about OpenSCAD is that it makes it easy to 3D model things which... | Hacker News, accessed February 18, 2026, https://news.ycombinator.com/item?id=46338565

[35]The great thing about OpenSCAD is that it makes it easy to 3D model things which... | Hacker News, accessed February 18, 2026, https://news.ycombinator.com/item?id=46338565

[36]Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

or the slicer, the tool will fail or produce cryptic errors. In a school environment with restricted user permissions, setting up the necessary environmental variables can be a significant administrative hurdle.[37]

The Risks of Automated Scripting

The power of 3DMake lies in its ability to string together actions: 3dm build slice print.[38] This "one-command" fabrication is efficient but dangerous if the student bypasses visual verification. If the OpenSCAD code contains a subtle error that results in a "non-manifold" mesh, the slicer may still produce a G-code file that causes the printer to behave erratically-such as extruding into mid-air.24

Limitation	Technical Root Cause	Educational Impact
CLI Barrier	No graphical interface for configuration.	Steep learning curve for students with zero terminal experience.[39]
API Dependency	AI features require external internet and API keys.	Advanced features fail in offline school networks.[40]
Slicer Lock-in	Reliant on external templates for G-code generation.	Students may not learn the nuance of slicing settings.[41]
Feedback Latency	No real-time "live" preview in the editor.	The "edit-compile-view" cycle is slower than GUI-based CAD.[42]

Non-Manifold Geometry and Slicing Errors

A recurring challenge for 3DMake users is the production of "non-manifold" STLs. A manifold object is "water-tight"-it has a clear inside and outside.[^28] OpenSCAD can easily produce non-manifold geometry through "zero-thickness" walls or improperly closed polyhedrons.[^24] 3DMakes build command will generate the STL without warning, but the slice command may then fail or produce

[37] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[38] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[39] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[40] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[41] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

[42] Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. Retrieved from https://github.com/tdeck/3dmake

a corrupt G-code file.[^24] This requires the student to learn mesh verification skills, often requiring third-party tools like MeshLab or PrusaSlicers repair functions.[^28]

Local Resources and Community Support: Salt Lake County Ecosystem

For students located in the Salt Lake County area, the transition from digital model to physical object is supported by a robust network of makerspaces and public library resources.

Public Library "Create Spaces" and Creative Labs

The Salt Lake County and City Library systems offer specialized makerspaces where students can bring their 3DMake-generated files for printing.

Facility	Location	Key Hardware	Policies/Costs
SLC Public Creative Lab	Main Library (Level 1)	Prusa i3 MK3, LulzBot Taz 5, Elegoo Mars 2	Free for prints under 6 hours; Material cost + $0.50/hr otherwise.[^31]
County Library "Create"	Daybreak, Granite, Kearns, etc.	Flashforge Adventurer 5M Pro, LulzBot Workhorse	$0.06 per gram of filament used.[^26]
Make Salt Lake	663 W 100 S, SLC	CNC, Metal Shop, Large-scale FDM and Resin	Membership-based; offers certification classes for advanced tools.[^32]

Higher Education and Specialized Maker Hubs

For advanced students, the University of Utah provides several makerspaces, including the Lassonde Studios and the Eccles Health Sciences Library Technology Hub.[^34] These centers offer access to "industrial-grade" printing technologies, such as Selective Laser Sintering (SLS), which require even more rigorous safety training regarding inert gas (Argon/Nitrogen) asphyxiation hazards.[43]

Conclusion: The Pedagogy of Programmatic Manufacturing

The integration of OpenSCAD and 3DMake into a high school curriculum is a powerful strategy for developing the next generation of engineers. By shifting the focus from "visual sculpting" to "mathematical definition," students are

[43] 3D Printer Safety - Environmental Health & Safety - University of Tennessee, Knoxville, accessed February 18, 2026, https://ehs.utk.edu/index.php/table-of-policies-plans-procedures-guides/3d-printer-safety/

forced to confront the underlying logic of their designs. The 3DMake toolchain facilitates this by removing the friction of manual rendering and slicing, allowing the student to stay in the ”flow state” of coding.
However, the success of this instructional model depends on a comprehensive understanding of its limitations. The instructor must balance the efficiency of AI-assisted verification with a healthy skepticism of LLM spatial reasoning. They must enforce rigid safety protocols to mitigate the invisible risks of UFP and VOC emissions. And finally, they must guide the student through the idiosyncratic challenges of the OpenSCAD language-its absolute coordinates and its strict manifold requirements.

References

Deck, T. (2025). *3DMake: A command-line tool for 3D printing workflows*. GitHub. https://github.com/tdeck/3dmake
Gohde, J., & Kintel, M. (2021). *Programming with OpenSCAD: A beginner's guide to coding 3D-printable objects*. No Starch Press.
Gonzalez Avila, J. F., Pietrzak, T., & Casiez, G. (2024). *Understanding the challenges of OpenSCAD users for 3D printing*. Proceedings of the ACM Symposium on User Interface Software and Technology.
Google. (2025). *Vertex AI Gemini 3 Pro Preview: Getting started with generative AI*. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3
National Institute for Occupational Safety and Health (NIOSH). (2024). *Approaches to safe 3D printing: A guide for makerspace users, schools, libraries, and small businesses*. https://www.cdc.gov/niosh/blogs/2024/safe-3d-printing.html
Ohio State University Environmental Health and Safety. (2026). *3D printer safety concerns and ventilation*. https://ehs.osu.edu/kb/3d-printer-safety
Salt Lake City Public Library. (2026). *Creative Lab: Available hardware and 3D printing procedures*. https://services.slcpl.org/creativelab
Salt Lake County Library. (2026). *Create Spaces: Hardware specifications and filament fees*. https://www.slcolibrary.org/what-we-have/create
University of Utah. (2026). *Marriott Library ProtoSpace and Maker hubs*. https://lib.utah.edu/protospace.php
Washington State Department of Health. (2026). *3D printer and filament selection for safe school environments*. https://doh.wa.gov/community-and-environment/schools/3d-printers

Supplemental Learning Resources

This introduction is complemented by comprehensive textbooks and code examples:

- Programming with OpenSCAD: A Beginner's Guide to Coding 3D-Printable Objects - Complete reference covering OpenSCAD syntax, geometry con-

cepts, design patterns, and best practices
- Simplifying 3D Printing with OpenSCAD - Focused guide to practical workflows, optimization techniques, and real-world printing applications
- CodeSolutions Repository - Working OpenSCAD examples organized by topic and difficulty level, demonstrating all concepts covered in this curriculum
- Practice Worksheets and Guides - Printable materials for visualization practice, decomposition exercises, and conceptual assessment

Works cited

3D Make Foundation: Complete Curriculum Guide

A comprehensive, hands-on, screenreader accessible introduction to programmatic 3D design using OpenSCAD and 3DMake

Overview

This 11-lesson curriculum + 4 reference appendices teaches non-visual 3D modeling, parametric design principles, and the complete 3D printing workflow through integrated projects. Each lesson builds on prior knowledge, introducing real-world examples and hands-on activities.

Target Audience: High school and early undergraduate students, makers, and anyone interested in programmatic CAD and 3D printing

Estimated Total Time: `30-40` hours (11 lessons + 4 appendices + projects)

Contents:

- 11 progressive lessons (Foundations -> Leadership)
- 9 hands-on projects (integrated into lessons)
- 4 comprehensive reference appendices (1,`200`-1,`500` lines each)
- Complete learning paths for different skill levels

Lesson Structure

Lesson 1: Environmental Configuration and the Developer Workflow

Duration: 2.5-3.5 hours | Level: Beginner

Learn to install and configure 3dMake, create a project scaffold, and run your first build. This lesson establishes the foundation for all subsequent work.

Key Topics:

- Installation and tool verification (`3dm`, `openscad`)
- Project structure (`src/`, `build/`, `3dmake.toml`)
- Parametric design philosophy

- The `3dm build` command

Hands-On Activity:

- Create a 3dMake project
- Write a simple parametric cube model
- Build and inspect the generated STL

Checkpoint: You can initialize a project, edit a parametric model, and generate an STL file

Lesson 2: Geometric Primitives and Constructive Solid Geometry

Duration: 60 minutes | Level: Beginner

Discover how to combine basic shapes (cube, sphere, cylinder) using boolean operations (union, difference, intersection). This lesson introduces the mathematical foundation of 3D modeling.

Key Topics:

- Primitive shapes: `cube()`, `sphere()`, `cylinder()`
- CSG operations: `union()`, `difference()`, `intersection()`
- The 0.001 offset rule for avoiding non-manifold geometry
- Low-resolution renders for faster debugging

Hands-On Activity:

- Build three simple CSG examples
- Diagnose and fix a failing boolean operation
- Validate geometry in a slicer

Checkpoint: You understand CSG operations and can diagnose common geometry issues

Lesson 3: Parametric Architecture and Modular Libraries

Duration: 2.5 hours | Level: Beginner+

Learn to create reusable modules and organize code into libraries. This lesson introduces the ”don't repeat yourself” (DRY) principle for code reuse.

Key Topics:

- Module definitions with parameters
- Derived calculations from parameters
- Library organization (`lib/` folder)
- Using external libraries (BOSL2)
- Low-resolution testing with `$fn`

Hands-On Activity:

- Create a parametric bracket module

- Build variants by changing parameters
- Move a module into a reusable library

Checkpoint: You can create parametric modules and organize code into libraries

Lesson 4: AI-Enhanced Verification and Multimodal Feedback

Duration: 2-2.5 hours | Level: Intermediate

Use `3dm info` to generate AI diagnostics and validate designs. This lesson covers verification workflows and the strengths/limitations of AI in design.

Key Topics:

- The `3dm info` command for model analysis
- AI-generated model descriptions
- Comparing AI suggestions to deterministic validation
- Privacy and governance considerations
- Prompt engineering basics

Hands-On Activity:

- Run `3dm info` on a sample model
- Compare AI suggestions to slicer analysis
- Document an AI-assisted design decision

Checkpoint: You can use AI tools to supplement your design validation

Lesson 5: Safety Protocols and the Physical Fabrication Interface

Duration: 2.5-3.5 hours | Level: Intermediate

Transition from digital design to physical printing. Learn safety procedures, environmental controls, and the complete print workflow.

Key Topics:

- Hierarchy of Controls (elimination, engineering, administrative, PPE)
- Chemical and particulate emissions from printing
- Pre-print environmental and equipment checks
- Post-print inspection and measurement
- Spool metadata and documentation

Hands-On Activity:

- Conduct a safety briefing
- Verify environmental controls
- Perform a pre-print checklist
- Monitor and measure a completed print

Checkpoint: You understand safety procedures and can safely conduct supervised prints

Lesson 6: Practical 3dm Commands and Text Embossing

Duration: 2.5-3.5 hours | Level: Intermediate

Master the key 3dm commands by building a practical project-a customizable keycap with embossed text. This lesson ties together design and verification.

Key Topics:

- `3dm describe`: Text-based model analysis
- `3dm preview`: Generating 2D tactile prints
- `3dm orient`: Analyzing optimal print orientation
- `3dm slice`: Generating G-code
- Text embossing with `linear_extrude()` and `text()`

Project: Parametric Cube Keycap

- Customize `key_size`, `letter`, and `emboss_depth`
- Generate 3+ variants (small, medium, large)
- Test emboss quality in slicer preview

Checkpoint: You can generate keycaps with embossed text and understand all major 3dm commands

Lesson 7: Parametric Transforms and the Phone Stand Project

Duration: 3-3.5 hours | Level: Intermediate+

Apply transforms (translate, rotate, scale) to build a multi-part assembly. This lesson covers spatial positioning and practical product design.

Key Topics:

- Transform operations: `translate()`, `rotate()`, `scale()`
- Minkowski sum for edge rounding
- Multi-part assemblies
- Parametric angle and position variations
- Testing and validation

Project: Parametric Phone Stand

- Design base, back, and lip components
- Create configurations for phones, tablets, documents
- Test orientation with `3dm orient`
- Validate angles and friction after printing

Checkpoint: You can design multi-part assemblies with positioned components

Lesson 8: Advanced Parametric Design and Interlocking Features

Duration: 90-120 minutes | Level: Advanced

Design tolerance-critical assemblies where parts snap together. This lesson covers precision manufacturing principles.

Key Topics:

- Tolerance and clearance management
- Stack-up and cumulative tolerances
- Interlocking rims and snap-fit connectors
- Chamfers for quality finishing
- Tolerance sensitivity testing

Project: Stackable Storage Bins

- Design bins with interlocking rims
- Test different `stack_clear` values
- Create variants: small, medium, large
- Add optional dividers
- Document tolerance data

Checkpoint: You understand tolerance management and can design stackable assemblies

Lesson 9: Automation and 3dm Workflows

Duration: 2.5-3.5 hours | Level: Advanced

Automate design workflows using shell scripts. This lesson teaches batch processing and continuous integration concepts.

Key Topics:

- Chaining 3dm commands with `&&`
- Shell script basics for batch processing
- Library management (`3dm lib`)
- Variant testing matrices
- Production build workflows

Activities:

- Create a batch build script
- Automate parameter variant testing
- Generate production build reports
- Learn library management

Checkpoint: You can automate design workflows and manage parametric variants at scale

Lesson 10: Hands-On Practice Exercises and Troubleshooting

Duration: 5-6 hours | Level: Advanced

Complete integrated projects and learn to diagnose and fix common issues. This capstone lesson synthesizes all prior learning.

Exercise Sets:

Set A: Guided Projects

1. Phone Stand (Beginner): Parametric design with weight constraints
2. Keycap Set (Intermediate): Family of 5+ customizable caps
3. Storage System (Advanced): Stackable bins with tolerance management

Set B: Problem Diagnosis

- Non-manifold geometry detection and fixes
- Print failure prevention
- Dimensional accuracy troubleshooting

Set C: Validation & Documentation

- Design review checklists
- Troubleshooting documentation templates
- Quality assurance workflows

Checkpoint: You can complete real projects from concept to verified print

Learning Progressions

By Skill Level

Beginner Track (Lessons 1-3):

- Learn tools and project structure
- Understand geometric primitives and boolean operations
- Create your first parametric modules

Intermediate Track (Lessons 4-6):

- Explore verification and safety
- Master 3dm commands
- Build your first complete project (keycap)

Advanced Track (Lessons 7-10):

- Design complex assemblies
- Manage tolerances precisely
- Automate workflows and troubleshoot professionally

By Project Focus

Design & Modeling: Lessons 1-3, 7-8 Verification & Validation: Lessons 4, 6, 10 Safety & Printing: Lesson 5, 10 Automation: Lesson 9

Practice Exercises Quick Reference

Structured Exercises by Lesson

Lesson	Exercise Type	Deliverable
1	Setup & Configuration	Working project scaffold
2	Geometry Construction	3 CSG examples + fixes
3	Modular Design	Parametric bracket module
4	AI Verification	`AI-notes.md` with findings
5	Safety & Printing	Pre-print checklist + measurements
6	Command Mastery	3+ keycap variants
7	Multi-Part Assembly	Phone stand for 3+ devices
8	Tolerance Design	Stackable bins + tolerance matrix
9	Automation	Batch build script + variants
10	Integration	3 complete projects + documentation

Code Examples Repository

All code examples from lessons are provided as:

1. Inline OpenSCAD code blocks in each lesson (copy-paste ready)
2. Shell script examples for automation tasks
3. Documentation templates for reproducible workflows

Accessibility Features

This curriculum is designed for non-visual learners:

- Text descriptions of all models (`3dm describe`)
- Tactile 2D previews (`3dm preview`)
- Structured written documentation
- Command-line based (no graphical interface required)
- Parametric organization for clear understanding
- Measurement-based validation without visual inspection

Recommended Reading Order

Option 1: Linear (Complete Foundation)

Lesson 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10

Best for: First-time learners wanting comprehensive understanding

Option 2: Fast Track (Design Focus)

Lesson 1 -> 2 -> 3 -> 6 -> 7 -> 8 -> 9

Best for: Experienced designers new to programmatic CAD

Option 3: Project-Based

Lesson 1 -> 2 -> 3 -> 6 (Keycap Project) Then: Lesson 7 (Phone Stand) Then: Lesson 8 (Storage Bins) Then: Lesson 9 (Automation)

Best for: Learning by doing with integrated projects

Assessment and Completion

Lesson Completion Criteria

Each lesson includes:

- Learning objectives (concepts you'll understand)
- Step-by-step tasks (hands-on activities)
- Checkpoints (verification milestones)
- Quiz (10 self-assessment questions)
- Extension problems (10 advanced challenges)

Project Completion Tracking

Project	Lessons	Output Files	Est. Print Time
Keycap Set	1-6	5+ SCAD files, 3+ STL files	15-30 min total
Phone Stand	1-7	1 SCAD with 3+ variants	30-60 min total
Storage Bins	1-8	1 parametric SCAD, 3 sizes	60-90 min total

Troubleshooting Guide

Common Issues and References

Issue	Related Lesson	Solution
Model won't build	Lesson 2	Check for non-manifold geometry with `3dm describe`
Parts don't fit	Lesson 8	Use tolerance testing matrix to find correct `stack_clear`
Print fails	Lesson 5	Verify pre-print checklist and slicer settings
Dimensions off	Lesson 10	Conduct tolerance sensitivity study and apply correction factor
Slow renders	Lesson 3	Lower `$fn` for faster debugging
Script errors	Lesson 9	Use `&&` to chain commands properly; check file paths

Next Steps After Completion

Upon completing this curriculum, you're ready for:

1. Advanced 3D Design: Explore BOSL2 library features, parametric assemblies
2. Multi-Material Printing: Design parts for different materials (TPU, Nylon, etc.)
3. Functional Design: Create mechanical assemblies with bearings, gears, mechanisms
4. Community Contribution: Share your designs and libraries with OpenSCAD community
5. Professional Applications: Apply skills to product design, prototyping, manufacturing

Resources and References

Official Documentation

- OpenSCAD Manual
- 3DMake GitHub
- BOSL2 Library

Printers & Slicers

- PrusaSlicer Documentation
- Bambu Studio Documentation
- OrcaSlicer Documentation

3D Printing Guides

- Prusa3D Troubleshooting
- All3DP 3D Printing Guide

Community

- OpenSCAD Forums
- Prusa Community
- Reddit r/3Dprinting

Lesson 11: Stakeholder-Centric Design and the Beaded Jewelry Project

Duration: 3.5-4.5 hours | Level: Advanced/Leadership

The final lesson teaches you to design for real users, not just yourself. You'll learn to conduct stakeholder interviews, extract functional requirements, and iterate based on real feedback. This bridges the gap between maker and design professional.

Key Topics:

- Stakeholder identification and interview techniques
- Open-ended questioning strategies
- Extracting functional requirements from interview data
- Defining measurable acceptance criteria
- Design iteration based on user feedback
- Documentation for reproducibility and accessibility

Project: Beaded Jewelry Bracelet Holder

- Interview an actual stakeholder (or use provided scenario)
- Extract their specific needs and constraints
- Design a parametric holder based on their requirements
- Test with actual bracelets and iterate based on feedback
- Document the complete design process

Checkpoint: You can conduct a real interview, translate needs into measurable requirements, and iterate a design based on user feedback

3D Make Foundation Appendices

Appendix A: Comprehensive Slicing Guide - All Major Slicers

A complete reference covering multiple major slicers (PrusaSlicer, Bambu Studio, Cura, SuperSlicer, OrcaSlicer, IdeaMaker, Fusion 360 built in slicer).

What You'll Find:

- Setup and workflow for each slicer
- Accessible parameter explanations (non-visual)
- Command-line usage for PowerShell integration
- Troubleshooting guide for common problems
- Comparison table for choosing the right slicer

When to Use: Reference whenever slicing, switching slicers, or troubleshooting print quality

Appendix B: Material Properties & Selection Guide

Comprehensive reference for 6 common filament materials (PLA, PETG, ABS, TPU, Polycarbonate, Nylon).

What You'll Find:

- Properties table (strength, flexibility, temperature, cost)
- Ideal projects and use cases for each material
- Printing parameters and settings

- Quality factors and how to verify filament quality
- Storage and maintenance
- Cost analysis and brand recommendations
- Material selection decision tree

When to Use: Reference when choosing material for a project or troubleshooting print quality issues

Appendix C: Tolerance Testing & Quality Assurance Matrix

Measurement-based QA methodology designed to be used non-visually (with calipers, scales, and functional tests).

What You'll Find:

- Essential measurement tools (digital calipers, scales, go/no-go gauges)
- Step-by-step measurement procedures for all common dimensions
- Functional testing methods (load testing, assembly testing, durability testing)
- Tolerance stack-up calculations for multi-part designs
- Troubleshooting guide for common dimensional problems
- QA checklist template
- Accessibility-focused best practices

When to Use: Reference when starting a new project (create test plan), after printing (verify dimensions), or when troubleshooting quality issues

Appendix D: PowerShell Integration for SCAD Workflows

Shows how to automate 3D design workflows using PowerShell scripts (from PowerShell_Foundation Lessons 1-6).

What You'll Find:

- Basic workflow automation (SCAD -> STL -> G-code)
- Parametric sweep (test 5, 10, or 100 design variations automatically)
- Batch build for multiple files
- Print logging and quality tracking
- Printer communication (USB transfer, network monitoring)
- Complete full workflow integration
- PowerShell skills mapped to SCAD applications
- Accessibility considerations and best practices

When to Use: Reference when automating repetitive tasks, testing parameter variations, or building a batch of designs

Appendix E: Advanced OpenSCAD Concepts

Optional topics for experienced designers tackling specialized applications. Five in-depth modules with complete working examples.

What You'll Find:

1. Gears and Mechanical Components
    - Gear terminology and tooth geometry
    - Involute gear algorithm
    - Parametric servo gearbox design
    - Belt and pulley systems
2. Batch Processing and Statistical Analysis
    - Automated parameter sweep generation
    - Statistical summary scripts
    - Data-driven design selection
    - Comparison and analysis frameworks
3. Performance Optimization
    - Render time measurement and profiling
    - Resolution parameter strategy
    - Caching complex calculations
    - Preview vs render optimization strategies
4. Print Orientation and Support Structure Algorithms
    - Strength orientation analysis
    - Support material minimization
    - Bridge span calculations
    - Optimal slicing parameter determination
5. Recursive Function Patterns
    - Basic recursion with base cases
    - Fractal generation
    - Nested component assembly
    - Performance considerations for deep recursion
    - Practical cable management applications

When to Use:

- When working with mechanical systems requiring gears or synchronized motion
- When testing design variations systematically
- When optimizing complex models for faster iteration
- When designing for specific print orientations or minimizing support
- When building fractal or deeply nested structures
- For advanced library and framework development

Accessibility: Each example includes comprehensive comments, starts with simplified versions before advanced techniques, and documents performance implications for both visual and non-visual users

Curriculum Revision History

Version	Date	Changes
2.1	Feb 2026	Added Appendix E (Advanced OpenSCAD); Enhanced Lessons 3, 6, 7, 8, 9 with advanced topics
2.0	Feb 2026	Added Lesson 11 (Stakeholder Design) + 4 Appendices; Consolidated Units 0-3 content
1.0	Feb 2026	Initial comprehensive curriculum with 10 lessons + 5 projects

Feedback and Contributions

Have suggestions to improve this curriculum? Found an issue in a lesson or code example?

Please reach out with:

- Lesson number and specific content reference
- Description of the issue or suggestion
- Proposed solution (if applicable)

Happy designing!

3dmake: Non-Visual 3D Printing Tutorial

Estimated time: 45-75 minutes

Learning Objectives

- Describe the 3dMake command-line workflow and project structure
- Create a new 3dMake project, edit `src/main.scad`, and run `3dm build`
- Slice a model and produce a tactile preview or full G-code

Materials

- Computer with 3dMake installed
- Screen reader (NVDA, JAWS, Orca) configured
- Example project files in `assets/` or classroom repo

Step-by-step Tasks

1. Verify prerequisites: confirm `3dm`, `openscad`, and your slicer are discoverable in the terminal (`which 3dm`, `which openscad`).
2. Create a new project: `3dm new` -> open `src/main.scad` with `3dm edit-model` and add a simple cube: `cube([20,20,20]);`.
3. Build the project: `3dm build` and confirm `build/main.stl` exists.
4. Slice a preview: `3dm preview slice` then `3dm preview print` (or export preview STL) to test tactile output.
5. Slice for full print: `3dm build slice` -> inspect `build/main.gcode` with a layer-preview option in your slicer.

Checkpoints

- After step 2 you can open and edit `src/main.scad` from the terminal.
- After step 3 the `build/` folder contains `main.stl`.

Quick Quiz (5)

1. What command creates a new 3dMake project?
2. Where does 3dMake put compiled STLs by default?
3. How do you open the model editor from the CLI?
4. What command slices a preview for tactile testing?
5. Why is it useful to run a preview before a full print?

Extension Problems (10)

1. Create two variants of a parameterized cube (different sizes) and export both STLs; compare their file sizes and estimated print times.
2. Add a `3dmake.toml` overlay that changes layer height and document the visible effect on the preview.
3. Build and slice a small object, then import the STL into a second slicer and report any differences in estimated time.
4. Create a short shell script that automates new -> edit -> build for a classroom scaffold.
5. Describe three safety checks you will perform before starting a multi-hour print.
6. Develop an accessibility audit of 3DMake's web interface and CLI tools; test keyboard navigation, screen-reader compatibility, and error message clarity.
7. Build a 3DMake workflow automation script: integrates model creation, parameter validation, and batch STL generation.
8. Create a 3DMake best practices guide for your classroom: document common patterns, troubleshooting tips, and performance optimization.
9. Design a parametric model library in 3DMake with shared modules; test reuse across multiple student projects.

10. Compare 3DMake vs. desktop slicers on 5+ models; create a decision matrix for when to use each tool.

Notes: This lesson is intended to be hands-on. If networked AI features are not configured, skip the AI verification steps and focus on deterministic renders and slicer previews.

Helpful Commands

Library commands

```
3dm list-libraries
3dm install-libraries
```

Help and version

```
3dm help
3dm version
```

References (APA)

Deck, T. (n.d.). *3dmake: Non-visual 3D design and 3D printing tool*. GitHub. Retrieved February 18, 2026, from http://github.com/tdeck/3dmake

OpenSCAD. (n.d.). *The programmers solid 3D CAD modeller*. Retrieved February 18, 2026, from https://openscad.org

```


Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See [https://example.com](https://example.com) and [https://example.com](https://example.com) for vendor documentation.
```

Lesson 1: Environmental Configuration and the Developer Workflow

Estimated time: 90-120 minutes

Learning Objectives

- Install and verify 3dm[44], openscad[45], and a slicer are discoverable in the terminal

[44] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[45] OpenSCAD Manual - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual

- Initialize a 3dMake project and understand the project scaffold (`src/`, `build/`, `3dmake.toml`)
- Edit `src/main.scad` using OpenSCAD's parametric design capabilities[46], run `3dm build`, and inspect the generated `build/main.stl`

Materials

- Terminal with 3dMake installed
- Editor (VS Code or Notepad)
- Example scaffold or classroom repository

Step-by-step Tasks

1. Run `./3dm setup` or follow instructor's installation notes; confirm tools with `which 3dm` and `which openscad`. Verify your 3dMake installation is properly configured[47].

2. Create a project scaffold with `3dm new` and open `src/main.scad` using `3dm edit-model`. For a comprehensive introduction to the workflow, consult the OpenSCAD documentation[48].

3. Add three top-level parameters (e.g., `width`, `height`, `thickness`) and a minimal model. This demonstrates the parametric design philosophy central to OpenSCAD[49].

   Example `src/main.scad`:

```
// Top-level parameters (change these to customize your model)
width = 50; // mm
height = 30; // mm
thickness = 5; // mm
// Main model
cube([width, height, thickness]);
```

4. Run `3dm build` and verify `build/main.stl` exists. Compare this build process to standard OpenSCAD workflows[50].

5. Open the STL in your slicer to check for thin walls or non-manifold geometry[51]; if issues appear, iterate on `main.scad` and rebuild.

[46]OpenSCAD Parametric Design - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Variables
[47]3DMake GitHub Repository - https://github.com/tdeck/3dmake
[48]OpenSCAD Manual - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual
[49]OpenSCAD Parametric Design - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Variables
[50]OpenSCAD Review - Worth learning? - CadHub, accessed February 18, 2026, https://learn.cadhub.xyz/blog/openscad-review/
[51]OpenSCAD Prompt Creation - DocsBot AI, accessed February 18, 2026, https://docsbot.ai/prompts/technical/openscad-prompt-creation

6. Try modifying the parameters and running `3dm build` again to see how parametric design allows you to quickly create variants.

Code Style and Documentation Standards

Professional OpenSCAD code follows consistent documentation practices for clarity, maintainability, and accessibility. As you write more complex designs, good documentation becomes essential for both you and anyone reading your code.

Comment Types and When to Use Them

File Header Comments - Describe the entire file's purpose:

```
// ============================================================
// Parametric Phone Stand - Design v2.1
// ============================================================
// Purpose: Multi-angle viewing stand for phones/tablets
// Author: Alex Chen
// Created: February 2026
// Last Modified: February 23, 2026
//
// Parameters: phone_width, angle, lip_height
// Dependencies: None (standalone file)
// Print Time Estimate: 45-60 min (PLA, 0.20mm layer height)
// Material: ~50g PLA
// ============================================================
```

Section Comments - Organize code into logical blocks:

```
// ==========================================
// CUSTOMIZABLE PARAMETERS
// ==========================================
phone_width = 75; // mm
phone_height = 150; // mm
stand_angle = 60; // degrees
// ==========================================
// DERIVED CALCULATIONS (calculated from parameters)
// ==========================================
base_width = phone_width + 20;
lip_height = 15;
// ==========================================
// MODULE DEFINITIONS
// ==========================================
module base() { ... }
module stand() { ... }
```

Inline Comments - Clarify complex logic:

```
// Use minkowski for smooth edges (prevents sharp corners)
module rounded_base() {
  minkowski() {
    cube([width - 2*radius, depth - 2*radius, height], center=true);
    cylinder(r=radius, h=0.01, $fn=32); // $fn=32 for smooth curve
  }
}
```

Parameter Documentation - Explain units and constraints:

```
// Parameter ranges and units
width = 100; // mm - must be > 50 mm for stability
height = 50; // mm - can be 20-100 mm
wall_thickness = 2; // mm - 1.5-3 mm recommended (too thin breaks, too thick →
    ↪ wastes material)
angle = 45; // degrees - 30-80 degrees typical
$fn = 32; // render quality - use 16-20 for preview, 32+ for export
```

Documentation Best Practices

1. Write for Your Audience

- Document assumptions and constraints
- Explain non-obvious design choices
- Use clear parameter names (`wall_thickness` not `w`)

2. Include Module Documentation

```
// Create a hollow box with walls of specified thickness
// Parameters: outer_w, outer_d, outer_h (mm), wall (mm)
// Example: hollow_box(50, 50, 50, 2) creates 50x50x50 box with 2mm walls
module hollow_box(outer_w, outer_d, outer_h, wall) {
  difference() {
    cube([outer_w, outer_d, outer_h], center=true);
    cube([outer_w - 2*wall, outer_d - 2*wall, outer_h], center=true);
  }
}
```

3. Document Known Limitations

```
// Phone Stand v2
// Limitations:
// - Not suitable for phones heavier than 200g
// - Recommend PLA or PETG (TPU may warp at recommended angles)
// - Print with 20%+ infill for stability
// - Supports may be needed on underside of angle > 70 degrees
```

4. Accessibility in Code Documentation

- Use plain language, not jargon (or explain jargon)
- Explain visual concepts in text (e.g., ”fillet radius of 3mm rounds sharp corners”)
- Include parameter ranges and units always
- Describe geometry relationships (e.g., ”lip height 1/3 of stand height”)

Example: Well-Documented File

```
// ============================================================
// Parametric Keycap with Embossed Letter
// ============================================================
// Purpose: Customizable keyboard keycap for 3D printing
// Applications: Custom keyboards, gaming, accessibility
// Print Time: 3-5 min per cap (varies by size)
// Material: ~2g PLA per cap
//
// PARAMETERS TO CUSTOMIZE:
// - cap_size: 12-28 mm (small to large keycap)
// - cap_height: 6-15 mm
// - letter: any single character to emboss
//
// PRINT RECOMMENDATIONS:
// - Layer height: 0.15 mm (better letter quality)
// - Infill: 15% (sufficient for keyboard use)
// - No supports needed
// - Print with smooth base facing bed
//
// TESTING CHECKLIST:
// - Measure cap_size with calipers (should match design within 0.3mm)
// - Test embossed letter is legible (raised ~0.5mm from surface)
// - Test fit on actual keyboard switch (should snap fit snugly)
// ============================================================
// ===========================================
// CUSTOMIZABLE PARAMETERS
// ===========================================
cap_size = 14; // mm - key size (typical: 12-18 for alphanumeric)
cap_height = 10; // mm - distance from base to top
wall_thickness = 1.2; // mm - side wall thickness (1-1.5 typical)
letter = "A"; // Character to emboss on top
emboss_depth = 0.8; // mm - how deep letter is raised (0.5-1.0 recommended)
// ===========================================
// DERIVED CALCULATIONS
// ===========================================
inner_size = cap_size - 2 * wall_thickness;
// ===========================================
// MODULE: Hollow shell
// ===========================================
module keycap_shell() {
  difference() {
    // Outer box
    cube([cap_size, cap_size, cap_height], center = false);
    // Hollow interior (removed part)
    translate([wall_thickness, wall_thickness, wall_thickness])
      cube([inner_size, inner_size, cap_height], center = false);
  }
}
// ===========================================
// MODULE: Embossed letter on top
// ===========================================
module embossed_letter() {
  translate([cap_size / 2, cap_size / 2, cap_height - 0.01])
    linear_extrude(height = emboss_depth)
      text(letter,
           size = cap_size * 0.5,
           halign = "center",
           valign = "center",
           font = "Impact:style=Regular");
}
// ===========================================
// ASSEMBLY: Combine shell + emboss
// ===========================================
union() {
  keycap_shell();
  embossed_letter();
}
```

This level of documentation makes your code:

- Reusable: Others can use it without reading every line
- Maintainable: You can modify it months later and understand why things are designed a certain way
- Accessible: Non-visual users can understand the design intent and constraints
- Professional: Employers and collaborators trust well-documented code

Checkpoints

- After step 2 you can locate `3dmake.toml` and the `build/` directory. Ensure your project scaffold matches the expected structure described in the 3dMake repository[52].
- After step 4 the `build/` folder contains a valid `main.stl`. Verify the geometry using your slicer's validation tools[53].
- After this section, your code includes file header, parameter documentation, and module descriptions following professional standards.

Understanding 3D Printing Technology: The FDM Pipeline

Before you send your first print, it's important to understand how 3D printers work and how the choices you make in the slicer affect the final result. This section builds on the workflow you learned above by explaining *what happens after* `3dm build`.

The FDM (Fused Deposition Modeling) Process

FDM printing builds objects layer by layer, where each layer is a thin horizontal slice of your STL file. Here's the complete pipeline:

1. STL File -> Your 3dMake model exported as an STL geometry file
2. Slicer Analysis -> Software like PrusaSlicer reads the STL and divides it into layers
3. G-code Generation -> The slicer converts layers into machine instructions (coordinates, temperature, speed)
4. Printing -> The printer reads G-code, heats the nozzle to ~200-230C, and extrudes plastic one layer at a time
5. Cooling & Solidification -> Each layer cools and bonds to the layer below

Critical Settings That Affect Your Print

When you open your STL in a slicer, you'll encounter several parameters that directly impact quality, time, and strength:

Layer Height

- Definition: The thickness of each printed layer (typically 0.15-0.30 mm)
- Effect on Time: Smaller layers = more detail but longer print time. A layer height of 0.15 mm prints slower than 0.30 mm because more layers must be printed
- Effect on Quality: Smaller layers produce smoother surfaces; larger layers print faster but appear more ”stepped”

[52] 3DMake GitHub Repository - https://github.com/tdeck/3dmake

[53] Slicer Validation Tools - PrusaSlicer Documentation - https://docs.prusa3d.com/en/guide/39012-validation-tools/

- Common Choice: 0.20 mm is a good balance for classroom projects

Infill

- Definition: The interior solid percentage of your model (0-100%)
- Purpose: Infill provides internal strength without using solid material throughout (which would be wasteful and heavy)
- Common Values for Classroom: 15-20% infill is typical; 10% for very light parts, 50% for functional parts
- Infill Patterns: Grid, gyroid, or honeycomb patterns determine how the internal structure looks. Grid is simple and fast; gyroid is strong but more complex
- Rule of Thumb: Higher infill = stronger, heavier, and longer print time

Supports

- Definition: Temporary structures the printer creates to hold overhanging geometry during printing
- When Needed: Any geometry that "hangs" at a steep angle (typically > 45 from vertical) requires supports
- Post-Processing: Supports must be removed after printing (breaking them off, dissolving them, or picking them away)
- Cost: Supports increase print time and waste material, so good STL design minimizes them

Why This Matters for Your Design

When you wrote `src/main.scad` in the tasks above, you created a parametric model. Those parameters become *constraints* that affect how well your part prints:

- A `thickness` of 0.5 mm might be too thin for FDM (will break easily)
- A `width` of 300 mm will take many hours to print
- Sharp corners and thin walls can cause printing failures

Understanding the FDM pipeline helps you design parts that not only look correct in OpenSCAD but will actually *print successfully*.

Next Steps: The Slicer

After you create an STL with `3dm build`, you open it in a slicer to:

1. Verify geometry (check for thin walls or non-manifold faces)
2. Set layer height, infill, and supports
3. Preview the layers to see what the printer will do
4. Export as G-code to send to the printer

You'll practice this workflow in Lesson 2, where you'll iterate on a simple part and resolve common printing issues.

Getting Started: Guided Projects & Extension Resources

Once you've completed this lesson, you're ready to work on hands-on projects. The curriculum includes several guided projects and extension resources:

Extension Projects (Beginner to Advanced)

These projects are located in the Lesson 1 assets folder and are designed to reinforce your skills in practical contexts:

- Your First Print (Lesson 1 Assets - Your First Print)
  - Goal: Low-friction introduction to the complete printing workflow
  - Skills: Setup, basic slicing, first-time print validation
  - Best for: After completing Lesson 1
  - Asset folder: assets/Lessons_3dMake_1/Your_First_Print/
- Basic Project Scaffold Template (Lesson 1 Assets - SCAD Template)
  - A starter template for your own 3D printing projects
  - Includes parameter configuration and TODO sections

Learning Series Sample Projects

The `3dmake_learning_series/` folder contains worked examples aligned with this curriculum:

- 01_cube_keycap (Beginner) - Text embossing basics
- 02_parametric_phone_stand (Intermediate) - Transforms and Minkowski fillets
- 03_stackable_bins (Advanced) - Tolerance and assemblies

Reference Materials

Quick-reference guides are available in `Reference_Materials/`:

- 3dmake-setup-guide.md - Complete setup walkthrough and command reference
- openscad-cheat-sheet.md - Keyboard shortcuts, syntax, and common functions
- filament-comparison-table.md - Material properties for different print scenarios
- master-rubric.md - Assessment criteria for evaluating student work
- markdown-starter-guide.md - Documentation best practices

Quiz - Lesson 3dMake.1 (10 questions)

1. What command initializes a 3dMake project?
2. What folder holds generated STLs?
3. How do you open the main model editor from the CLI?
4. Why is it useful to run `3dm build` frequently during development?
5. Give one reason to prefer an external editor over editing inline.
6. True or False: 3dMake requires a graphical user interface to use effectively.
7. Explain what the `3dmake.toml` file does in your project.
8. Describe what the `src/`, `build/`, and other project scaffold folders are used for.
9. How would you compare the 3dMake build workflow to traditional OpenSCAD workflows?
10. What validation steps should you perform after running `3dm build` and before sending a file to print?

Extension Problems (10)

1. Add a README entry explaining your top-level parameters and expected units. Reference best practices from the OpenSCAD documentation[54].
2. Create a parameter variant by changing `width` by 20% and build both variants; compare dimensions with calipers. This demonstrates the power of parametric design discussed in programming resources[55].
3. Script a `3dm` command sequence that automates new -> edit -> build for the scaffold. Review the 3dMake test suite for inspiration[56].
4. Intentionally create a thin-wall error and document the steps you took to find and fix it. Consult slicing guides[57] for identifying common geometry issues.
5. Prepare a short instructor sign-off checklist describing safety checks before printing.
6. Build a variant testing suite: create 5+ parameter combinations, export STLs, and compare file sizes and estimated print times.
7. Create a 3dMake project template with best-practice structure, documentation, and reusable modules for future projects.
8. Develop a screen-reader accessibility guide for 3dMake CLI commands and parameter syntax.
9. Design a parametric part library in 3dMake format; document all parameters, units, and example usage.

[54] OpenSCAD Manual - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual

[55] OpenSCAD Parametric Design - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Variables

[56] 3dmake/e2e_test.py at main - GitHub, accessed February 18, 2026, https://github.com/tdeck/3dmake/blob/main/e2e_test.py

[57] Slicing Guides and Common Geometry Issues - PrusaSlicer Documentation, accessed February 18, 2026, https://docs.prusa3d.com/en/

10. Write a comprehensive troubleshooting guide for common 3dMake build errors, with solutions and prevention tips.

Supplemental Resources

For deeper exploration of OpenSCAD and parametric design, consult these resources:

- Programming with OpenSCAD EPUB Textbook - Comprehensive reference with examples of parametric design, transformations, and modules
- CodeSolutions Repository - Working OpenSCAD code organized by topic, including 3D primitives and parametric examples relevant to Lesson 1
- OpenSCAD Quick Reference - Visual syntax guide and command reference

3dMake Setup & Workflow

This guide walks you through installing 3dMake, creating projects, and managing your workflow efficiently.

Installing 3dMake

Prerequisites Before installing 3dMake, ensure you have:

- Windows or Linux operating system (macOS is not currently supported)
- Terminal or PowerShell access
- Internet connection to download 3dMake
- At least 100 MB of free disk space

Platform Support Supported:

- Windows (32-bit and 64-bit)
- Linux (x86-64 architecture)

Not Supported:

- macOS - Currently no macOS version is available

Step-by-Step Installation

Step 1: Download 3dMake for Your Operating System Visit the 3dMake releases page and download the appropriate version:

Windows:

1. Go to 3dMake Windows Download
2. This downloads `3dmakewindows.zip` to your computer
3. Right-click the file and select "Extract All..." (or use your preferred extraction tool)

4. Remember where you extracted it (e.g., `C:\Users\YourName\3dmake` or `C:\Program Files\3dmake`)

Linux:

1. Go to 3dMake Linux Download
2. This downloads `3dmakelinux.tar.gz` to your computer
3. Extract it using: `tar -xzf 3dmakelinux.tar.gz`
4. Remember where you extracted it (e.g., `~/3dmake` or `/opt/3dmake`)

Step 2: Open Your Terminal Windows:

- Press `Win + X` and select ”Windows PowerShell” or ”Terminal”
- For screen reader users: Use `Alt + F2` and type `powershell` if needed

Linux:

- Open your terminal application (Terminal, Konsole, GNOME Terminal, etc.)

Step 3: Navigate to 3dMake Directory Navigate to where you extracted 3dMake:

Windows:

```
cd C:\Users\YourName\3dmake
```

Linux:

```
cd ~/3dmake
```

For screen reader users: Use `pwd` (print working directory) to confirm your location.

Step 4: Run the Setup Command From inside the 3dMake directory, run:

Windows:

```
.\3dm setup
```

Linux:

```
./3dm setup
```

Follow the prompts to configure:

- Your default printer profile (e.g., Prusa MK4, Bambu Lab)
- OctoPrint connection (if you use OctoPrint)
- AI integration (optional - for model descriptions)
- Preferred text editor

Step 5: Complete Installation After setup finishes, 3dMake will be available from any directory. You can now use the `3dm` command from your terminal.

Do not delete the original 3dMake directory where you extracted the files, as 3dMake needs to reference it.

Step 6: Verify Installation

```
3dm --version
```

You should see the installed version number. If you see an error, verify:

- You're in the 3dMake directory
- The extraction completed successfully
- Your terminal has access to the extracted files

Step 7: Get Help

```
3dm help
```

This displays all available 3dMake commands.

Creating and Managing Projects

What Is a 3dMake Project? A 3dMake project is a folder structure that organizes:

- src/ folder - contains your OpenSCAD (.scad) files
- build/ folder - stores outputs that 3dMake generates (STL files, sliced GCODE, etc.)
- 3dmake.toml file - project configuration and settings
- README.md - project documentation

Creating Your First Project

Step 1: Choose or Create a Project Directory Decide where you want your project. Examples:

Windows:

```
C:\Users\YourName\Documents\3d-projects\
C:\Users\YourName\Desktop\MyProject\
```

Linux:

```
~/3d-projects/
~/Documents/3d-projects/
```

Create the directory if it doesn't exist:

Windows:

```
mkdir C:\Users\YourName\Documents\3d-projects\FirstProject
```

Linux:

```
mkdir -p ~/3d-projects/FirstProject
```

Step 2: Navigate Into Your Project Directory

Windows:

```
cd C:\Users\YourName\Documents\3d-projects\FirstProject
```

Linux:

```
cd ~/3d-projects/FirstProject
```

For screen reader users: Use `pwd` (print working directory) to confirm you're in the right folder.

Step 3: Initialize a 3dMake Project

```
3dm new
```

This creates the project structure:

```
FirstProject/
+------ src/ (stores your .scad files)
+------ build/ (stores generated files: STL, GCODE, etc.)
+------ 3dmake.toml (project configuration)
+------ README.md (project documentation)
```

The `src/main.scad` file is created as a starting template.

Step 4: Verify Project Creation

Windows:

```
Get-ChildItem -Force
```

Linux:

```
ls -la
```

You should see the `src/`, `build/`, `3dmake.toml`, and `README.md` files listed.

Working With Your Project

Creating OpenSCAD Models

Step 1: Create or Edit OpenSCAD Files The main model file is `src/main.scad`. You can edit it directly using 3dMake:

```
3dm edit-model
```

This opens `src/main.scad` in your configured text editor.

To edit a different model (if you create additional .scad files):

```
3dm edit-model -m mymodel
```

This opens `src/mymodel.scad`.

Manual File Creation:

You can also create `.scad` files directly in the `src/` folder using your preferred text editor:

- Visual Studio Code
- Notepad++ (Windows)
- Gedit (Linux)
- Nano or Vim (terminal-based)

Step 2: Building Your Model Before exporting, build the model from your OpenSCAD code:

```
3dm build
```

This converts your OpenSCAD code into a 3D mesh (geometry). The output is `build/main.stl`.

For a different model:

```
3dm build -m mymodel
```

This creates `build/mymodel.stl`.

Step 3: Viewing Model Information To see statistics about your model:

```
3dm info
```

Output includes:

- Bounding box - dimensions (X, Y, Z in millimeters)
- Volume - cubic millimeters of material
- Face count - total triangles in the mesh
- Manifold status - whether model is watertight (printable)

Example output:

```
Volume: 1234.56 mm
Bounding Box: 50.0 x 40.0 x 30.0 mm
Faces: 2048
Manifold: Yes
```

Step 4: Previewing Your Model 3dMake can create flat ”tactile previews” of your model (fast-printing 2D silhouettes):

```
3dm preview slice
```

This generates a preview STL and slices it (ready to print in minutes).

To print the preview directly:

```
3dm preview print
```

Available preview types:

- `3sil` - Three silhouettes (front, left, top) - default
- `frontsil` - Front-facing silhouette only
- `topsil` - Top-down silhouette only

Change preview type:

```
3dm preview -v topsil print
```

Step 5: Slicing and Preparing for Print To slice your model (convert STL to GCODE for your printer):

```
3dm build slice
```

This creates both:

- `build/main.stl` - the 3D model
- `build/main.gcode` - sliced for your printer

Step 6: Printing Directly from 3dMake If your printer is connected via OctoPrint or Bambu Labs (LAN mode):

```
3dm build slice print
```

This builds, slices, and sends directly to your printer in one command.

Or simply:

```
3dm build print
```

(The `print` command automatically includes slicing)

Managing Multiple Projects

Switching Between Projects

Method 1: Navigate via Terminal

```
# Leave current project
cd ..

# Enter a different project
cd ../SecondProject
pwd # Verify you're in the right place
```

For screen reader users, always use `pwd` after navigating to confirm your location.

Method 2: Create a Projects Directory Structure Create a central folder for all projects:

```
3d-projects/
+------ FirstProject/
| +------ src/
| +------ build/
| +------ 3dmake.toml
+------ SecondProject/
| +------ src/
| +------ build/
| +------ 3dmake.toml
+------ README.md (project index)
```

Method 3: Use a Project Index Create a `README.md` in your `3d-projects/` folder to track all projects:

```
# My 3D Projects

## Project List

1. FirstProject - My initial test models
   - Location: `./FirstProject/src/`
   - Status: In progress
   - Latest model: `main.scad`

2. SecondProject - Parametric keychain designs
   - Location: `./SecondProject/src/`
   - Status: Complete
   - Latest model: `keychain.scad`

3. ThirdProject - Functional brackets for printing
   - Location: `./ThirdProject/src/`
   - Status: In progress
   - Latest model: `bracketassembly.scad`
```

Building from All Projects To build models from all projects:

Windows:

```
Get-ChildItem -Directory | ForEach-Object {
    cd $.FullName
    3dm build
    cd ..
}
```

Linux/Bash:

```
for project in */; do
    cd "$project"
    3dm build
    cd ..
done
```

Workflow Best Practices

File Naming Conventions Use clear, descriptive names for your `.scad` files:

Good:

- `cube5cm.scad`
- `parametricboxv2.scad`
- `bracketformotor.scad`
- `main.scad` (default project model)

Avoid:

- `test.scad`
- `model1.scad`
- `finalfinalFINAL.scad`

Understanding TOML Configuration Files What is TOML?

TOML (Tom's Obvious, Minimal Language) is a human-readable configuration file format. It's designed to be simple and clear, making it easy to read and edit in any text editor.

Basic TOML Formatting Rules:

1. Key-value pairs - Each setting has a name and value separated by an equals sign:

```
projectname = "My Project"
```

2. Strings use quotes - Text values must be wrapped in double quotes:

```
editor = "code"
printerprofile = "prusaMK4"
```

3. Numbers don't need quotes - Numeric values stand alone:

```
scale = 1.05
copies = 3
```

4. Boolean values are true or false - Lowercase, no quotes:

```
autostartprints = true
editinbackground = false
```

5. Arrays use square brackets - Lists of values separated by commas:

```
overlays = ["supports", "PETG"]
libraries = ["bosl", "braille-chars"]
```

6. One setting per line - Each configuration on its own line
7. Comments start with # - Use for notes (not processed):

```
# This is my default printer
printerprofile = "prusaMK4"
```

How 3dMake Uses TOML:

3dMake has two TOML configuration files:

- Global config (`defaults.toml`) - Settings for all your projects (run `3dm edit-global-config` to edit)
- Project config (`3dmake.toml`) - Settings specific to one project (located in your project root)

Project settings override global settings. For example, if your global config says `printerprofile = "prusaMK4"` but your project's `3dmake.toml` says `printerprofile = "bambu"`, the project setting wins.

Project Configuration (3dmake.toml) The `3dmake.toml` file in your project root contains project-specific settings:

```
projectname = "My Project"
modelname = "main"
printerprofile = "prusaMK4"
overlays = []
editor = "code"
```

Edit your project configuration:

```
3dm edit-global-config
```

This opens your configuration file in your text editor.

Version Control (Optional) If you're using Git, add a `.gitignore` to avoid committing large build files:

Create `.gitignore` in your project root:

```
build/
*.stl
*.gcode
*.png
.DSStore
pycache/
```

Then initialize Git:

```
git init
git add .
git commit -m "Initial project setup"
```

Backup Strategy Regularly backup your `src/` folder:

Windows:

```
Copy-Item -Path "src" -Destination "backups/src$(Get-Date -Format 'yyyy-MM-
    dd')" -Recurse
```

Linux:

```
cp -r src backups/src$(date +%Y-%m-%d)
```

Documentation Keep a `src/README.md` describing each model:

```
# Models in This Project

## main.scad
- Purpose: Primary design for this project
- Parameters: width, height, depth
- Last modified: 2026-02-20
- Notes: Standard model for printing

## alternate.scad
- Purpose: Alternative design variant
- Parameters: width, height, depth, wallthickness
- Last modified: 2026-02-15
- Notes: Experimental version with snap-fit features
```

Configuring Your Text Editor By default, 3dMake uses:

- Windows: Notepad
- Linux: Nano (or your EDITOR environment variable)

To use a different editor, edit your global configuration:

```
3dm edit-global-config
```

Add or modify the `editor` line. Examples:

Windows (Visual Studio Code):

```
editor = "code"
```

Windows (Notepad++):

```
editor = '''C:\Program Files (x86)\Notepad++\notepad++.exe'''
```

Linux (Visual Studio Code):

```
editor = "code"
```

Linux (Gedit):

```
editor = "gedit"
```

Troubleshooting

Common Issues

"3dm command not found" Cause: 3dMake isn't in your PATH or you haven't completed setup.

Solution:

1. Verify you extracted 3dMake and completed `3dm setup`
2. Restart your terminal completely (close and reopen)
3. Check that you're not inside the 3dMake directory when running commands
4. On Linux, ensure you're using `./3dm` if 3dMake isn't in your PATH yet

"No such file or directory: src/main.scad" Cause: You're not in a valid 3dMake project directory.

Solution:

Verify you're in the project directory:

```
pwd
ls -la
```

You should see `src/`, `build/`, and `3dmake.toml` files. If not, run:

```
3dm new
```

"OpenSCAD error: syntax error at line 5" Cause: Your `.scad` file has incorrect OpenSCAD syntax.

Solution:

1. Open your model file: `3dm edit-model`
2. Check the line number mentioned in the error
3. Verify correct OpenSCAD syntax (matching parentheses, semicolons, etc.)
4. Save and try building again: `3dm build`

"Model won't render or build" Checklist:

- ☐ File is saved with `.scad` extension
- ☐ File is in the `src/` folder
- ☐ File has valid OpenSCAD syntax
- ☐ Build output has a specific error message (check line number)

Debug mode:

```
3dm build --debug
```

This provides more detailed error messages.

Screen reader isn't reading 3dMake output clearly Solution: Use the `--debug` flag for more verbose output:

```
3dm build --debug
```

This logs each step, making it easier for screen readers to follow.

"Permission denied" error (Linux) Cause: 3dMake executable doesn't have run permissions.

Solution:

```
chmod +x 3dm
./3dm setup
```

Cannot connect to printer Cause: OctoPrint or Bambu printer settings not configured correctly.

Solution:

1. Verify printer is running and connected to network
2. Edit configuration: `3dm edit-global-config`
3. Check these settings:
   - `octoprinthost` - correct IP/URL
   - `octoprintkey` - valid API key
   - `printmode` - set to "octoprint" or "bambulan"
4. Test connection: `3dm build print` (without actually printing first)

Quick Reference

Essential Commands

Command	Purpose
`3dm --version`	Show installed version
`3dm help`	Display all available commands
`3dm setup`	Initial setup (run in extracted directory)
`3dm new`	Initialize a new project
`3dm build`	Build OpenSCAD model to STL
`3dm build slice`	Build and slice to GCODE
`3dm build print`	Build, slice, and send to printer
`3dm edit-model`	Open main.scad in text editor
`3dm edit-model -m name`	Open a specific model file
`3dm info`	Show model statistics
`3dm preview print`	Create and print a tactile preview
`3dm list-profiles`	Show available printer profiles

Command	Purpose
`3dm list-overlays`	Show available slicer overlays
`3dm edit-global-config`	Edit global settings
`3dm orient print`	Auto-orient and print model

Directory Structure Reference

```
YourProject/
+------ src/ <- Place .scad files here
+------ build/ <- Built STL, GCODE auto-save here
+------ 3dmake.toml <- Project settings
+------ README.md <- Project documentation
```

Configuration Options (3dmake.toml) Common settings you can modify:

```
projectname = "My Project"
modelname = "main" # Default model to build
printerprofile = "prusaMK4" # Your printer
overlays = ["supports"] # Default slicing overlays
editor = "code" # Text editor to use
```

For full configuration options, see the GitHub repository: https://github.com/tdeck/3dmake

Next Steps

Once you're comfortable with basic projects:

1. Learn parametric design - Make reusable models with variables
2. Explore modules - Organize code into reusable functions
3. Add libraries - Use pre-built OpenSCAD libraries (BOSL, etc.)
4. Optimize workflows - Chain commands (`3dm build slice print`)
5. Automate models - Use loops and conditionals in OpenSCAD
6. Collaborate - Use Git to share projects with others

For advanced topics, see:

- OpenSCAD Manual
- 3dMake GitHub Documentation
- BOSL Library Documentation

Sources

3dMake GitHub Repository. (2026). *3dMake - Non-visual 3D design and printing*. Retrieved from https://github.com/tdeck/3dmake

3dMake Documentation. (2026). *Terminal quick start guide*. Retrieved from https://github.com/tdeck/3dmake/blob/main/docs/terminalquickstart.md

OpenSCAD Community. (2025). *OpenSCAD user manual*. Retrieved from https://en.wikibooks.org/wiki/OpenSCADUserManual

Revarbat (Ed.). (2024). *BOSL - Belfry OpenSCAD Library*. Retrieved from https://github.com/revarbat/BOSL/wiki

VSCode Setup Guide

For use with NVDA or JAWS on Windows

This guide walks you through setting up Visual Studio Code (VSCode) as an accessible code editor for writing OpenSCAD files, with a task runner that automatically previews your `.scad` code in OpenSCAD whenever you save.

Why VSCode Instead of the OpenSCAD Editor?

The built-in OpenSCAD editor has inconsistent behavior with screen readers - focus can jump unexpectedly, and the editor sometimes stops being read after certain actions. VSCode is a mainstream code editor with strong, well-tested accessibility support for both NVDA and JAWS.

You write your code in VSCode. OpenSCAD runs in the background (or in a separate window) to render the preview. You never have to interact with the OpenSCAD editor itself.

Install Required Software

1.1 Install VSCode Download from: https://code.visualstudio.com/

During installation:

- Check "Add to PATH" - this is important for running VSCode from PowerShell
- Check "Register Code as an editor for supported file types"

1.2 Install OpenSCAD Download from: https://openscad.org/downloads.html

Use the installer version (not the portable version). After installing, confirm OpenSCAD is in your PATH by opening PowerShell and typing:

```
openscad --version
```

You should hear a version number. If you get an error, see the PATH setup section in the PowerShell Foundation guide.

1.3 Install the OpenSCAD VSCode Extension (Optional but Recommended)

1. Open VSCode
2. Press `Ctrl + Shift + X` to open the Extensions panel
3. Type `openscad` in the search box

4. Install ”OpenSCAD Language Support” - this adds syntax highlighting and keyword completion for `.scad` files

Configure the Task Runner

VSCode uses a file called `tasks.json` to define custom commands you can run from the keyboard. We'll set up a task that opens your current `.scad` file in OpenSCAD for preview whenever you press a key.

2.1 Open or Create Your Workspace Folder All your `.scad` files for a project should be in one folder. Open that folder in VSCode:

```
cd ~/Documents/OpenSCAD_Projects
code .
```

The `.` tells VSCode to open the current folder as a workspace.

2.2 Create the Tasks File

1. In VSCode, press `Ctrl + Shift + P` to open the Command Palette
2. Type `Tasks: Configure Task` and press Enter
3. Select ”Create tasks.json file from template”
4. Select ”Others”

This creates a `.vscode/tasks.json` file. Replace its entire contents with the following:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Preview in OpenSCAD",
      "type": "shell",
      "command": "openscad",
      "args": ["${file}"],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "presentation": {
        "reveal": "silent",
        "panel": "shared"
      },
      "problemMatcher": []
    },
    {
      "label": "Export STL",
      "type": "shell",
      "command": "openscad",
      "args": [
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}.stl",
        "${file}"
      ],
      "group": "build",
      "presentation": {
        "reveal": "always",
        "panel": "shared"
      },
      "problemMatcher": []
    }
  ]
}
```

Save the file with `Ctrl + S`.

2.3 Run the Preview Task With any `.scad` file open and focused:

- Press `Ctrl + Shift + B` to run the default build task (Preview in OpenSCAD)
- OpenSCAD will open and display your model
- Switch back to VSCode with `Alt + Tab` to keep editing

To export an STL:

1. Press `Ctrl + Shift + P`
2. Type `Tasks: Run Task`
3. Select "Export STL"
4. The `.stl` file will be saved in the same folder as your `.scad` file

NVDA Settings for VSCode

3.1 Recommended NVDA Settings Open NVDA Menu (`NVDA + N`) -> Preferences -> Settings:

Speech category:

- Symbol level: Most (so you hear brackets, semicolons, and other syntax characters)

Browse Mode category:

- Uncheck "Use browse mode on page load in web content" - not needed for VSCode

3.2 Useful NVDA + VSCode Keyboard Shortcuts

Action	Keys
Read current line	`NVDA + Up Arrow`
Read from cursor	`NVDA + Down Arrow`
Spell current word	`NVDA + Numpad 2` (twice quickly)
Move to next/previous line	`Up / Down Arrow`
Move by word	`Ctrl + Left / Right Arrow`
Move to start/end of line	`Home / End`
Move to start/end of file	`Ctrl + Home / End`
Select all	`Ctrl + A`
Toggle line comment	`Ctrl + /`
Go to line number	`Ctrl + G`, type number, Enter

Action	Keys
Find in file	`Ctrl + F`
Open file in workspace	`Ctrl + P`, type filename

3.3 Punctuation Level When reading code, you need to hear all punctuation - semicolons, brackets, parentheses, and commas are all part of OpenSCAD syntax.

In NVDA: `NVDA + N` -> Preferences -> Settings -> Speech -> Symbol level: Most

You can also toggle punctuation level on the fly: `NVDA + P` cycles through None, Some, Most, All.

JAWS Settings for VSCode

4.1 Virtual Cursor JAWS may try to activate Virtual/Browse mode in VSCode. If VSCode stops responding to arrow keys for navigation and starts reading the page as HTML, press `JAWS Key + Z` to toggle Virtual mode off. You want to be in Application mode (not Virtual mode) when using VSCode.

4.2 Recommended JAWS Settings

- Punctuation level: All - `JAWS Key + Shift + 2` cycles through levels. Set to All for code editing.
- Reading rate: Adjust with `Alt + Ctrl + Page Up / Page Down`
- Spell current line: `JAWS Key + Up Arrow` twice quickly

4.3 Useful JAWS + VSCode Keyboard Shortcuts

Action	Keys
Read current line	`JAWS Key + Up Arrow`
Read from cursor	`JAWS Key + Down Arrow`
Read current character	`JAWS Key + Numpad 5`
Move by word	`Ctrl + Left / Right Arrow`
Move to start/end of line	`Home / End`
Toggle comment	`Ctrl + /`
Go to line	`Ctrl + G`
Open command palette	`Ctrl + Shift + P`

Notepad++ as an Alternative

If VSCode is too complex to set up, Notepad++ is a simpler screen-reader-friendly option for editing `.scad` files.

Setup

1. Download from: https://notepad-plus-plus.org/
2. Install the OpenSCAD syntax highlighting plugin:
   - Go to Plugins -> Plugin Admin
   - Search for ”OpenSCAD” - install if available
   - Alternatively, download a UDL (User Defined Language) file from the OpenSCAD community and import it via Language -> User Defined Language -> Import

Running OpenSCAD from Notepad++ Use the Run menu (`F5`) to configure a custom command:

```
openscad "$(FULL_CURRENT_PATH)"
```

Name it ”Preview in OpenSCAD” and assign it a shortcut key (e.g., `Ctrl + F5`).

NVDA + Notepad++ Tips

- Punctuation level: set to Most or All
- Use `Ctrl + G` to go to a specific line
- Use `Ctrl + F` to find text
- The status bar at the bottom of Notepad++ announces line and column numbers - useful for finding where errors are

Workflow Summary

Here is the complete workflow from writing code to printing:

1. Open VSCode (or Notepad++) code ~/Documents/OpenSCAD_Projects

2. Open or create a .scad file Ctrl + P -> type filename

3. Write your OpenSCAD code

4. Preview in OpenSCAD Ctrl + Shift + B (VSCode) Ctrl + F5 (Notepad++)

5. If the shape looks right, export STL Run ”Export STL” task in VSCode OR: In OpenSCAD, press F6 then File > Export > Export as STL

6. Open PrusaSlicer, import the STL, slice, export G-code

7. Load G-code onto SD card or USB and print

Troubleshooting

OpenSCAD doesn't open when I run the task

- Confirm OpenSCAD is installed and in your PATH: `openscad --version` in PowerShell

- If not found, add the OpenSCAD install folder to your PATH (see PowerShell Foundation guide)

VSCode isn't being read by my screen reader

- For JAWS: Press `JAWS Key + Z` to toggle out of Virtual mode
- For NVDA: Make sure you are focused inside the editor panel, not a sidebar
- Try clicking directly in the editor area with the mouse once to confirm focus

I hear ”unlabeled” for some VSCode elements

- This is a known VSCode accessibility limitation for some UI panels
- Use keyboard shortcuts rather than trying to navigate by element - the editor itself reads well

My .scad file has an error but I can't find it

- OpenSCAD will display an error in its console - use `Alt + Tab` to switch to OpenSCAD and arrow through the console to read the error
- Errors always include a line number - use `Ctrl + G` in VSCode to jump to that line

References

Microsoft. (2024). *Visual Studio Code accessibility*. https://code.visualstudio.com/docs/editor/accessibility

NV Access. (2024). *NVDA user guide*. https://www.nvaccess.org/files/nvda/documentation/userGuide.html

OpenSCAD. (n.d.). *OpenSCAD documentation*. https://openscad.org/documentation.html

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

Navigating This Curriculum - mdBook Guide

This curriculum is published as a web book using mdBook. This page explains how to find what you need, navigate between chapters, and use the book with a screen reader.

What Is mdBook?

mdBook is a tool that turns a collection of Markdown files into a navigable web book - similar to an online textbook. Each lesson or document is its own page (chapter), and they are organized into sections visible in the sidebar table of contents.

You can access the book from any web browser on any device.

Basic Navigation

Sidebar Table of Contents The left side of the page contains a table of contents showing all chapters organized by unit and lesson. You can click or tap any chapter title to jump directly to it.

On a small screen (phone or tablet), the sidebar may be hidden. Look for a hamburger menu icon (three horizontal lines) to show it.

Arrow Navigation At the bottom of every page there are Previous and Next links that take you through the chapters in order. You can also use keyboard arrow keys:

- `Left Arrow` - go to the previous chapter
- `Right Arrow` - go to the next chapter

Search The mdBook search feature indexes all content across all chapters.

- Click the search icon (magnifying glass) in the top bar, or press `S`
- Type your search term
- Results appear as a dropdown list with the chapter name and a snippet of context
- Click any result to jump to that chapter; the matching term will be highlighted
- Press `Escape` to close the search panel

Other Keyboard Shortcuts

Key	Action
`S`	Focus the search box
`Escape`	Close search results
`Left Arrow`	Previous chapter
`Right Arrow`	Next chapter
`T`	Toggle the table of contents sidebar

Screen Reader Navigation

With NVDA (Chrome, Firefox, or Edge recommended) Reading the page:

- Use `Up / Down Arrow` to read line by line
- Use `H` to jump between headings - this is the fastest way to skim a long lesson
- Use `Ctrl + F` (browser Find) or `S` (mdBook search) to find specific content

Table of contents:

- The sidebar is a navigation landmark. Press `D` to move between landmark regions, or use `NVDA + F7` to open the Elements List and select "Landmarks" to navigate to the sidebar directly.
- Within the sidebar, arrow through the list of links and press `Enter` to follow one.

Code blocks:

- Code examples are marked up as `<code>` elements. NVDA will read them inline.
- Punctuation level should be set to Most or All to hear semicolons, brackets, and other syntax characters in code examples.
- To copy a code block: navigate to the code, press `Ctrl + A` to select all, or use the copy button if present.

Tips:

- `NVDA + P` to cycle punctuation level - do this before reading code blocks
- `NVDA + F7` -> Links list - useful for navigating between major sections quickly
- `H` key (headings navigation) is your best friend on lesson pages with many sections

With JAWS (Chrome or Edge recommended) Reading the page:

- Use `Up / Down Arrow` in virtual cursor mode to read line by line
- Press `H` to jump between headings
- Press `JAWS Key + F6` to get a list of all headings on the page

Table of contents:

- Press `R` to move between landmark regions to reach the sidebar
- Within the sidebar, `Tab` through the links or use `Up / Down Arrow`

Code blocks:

- Set punctuation to All before reading code: `JAWS Key + Shift + 2`
- JAWS reads code blocks as regular text - navigate through them line by line

Tips:

- `JAWS Key + F5` - links list
- `JAWS Key + F6` - headings list
- `Ctrl + F` - browser find, works alongside mdBook search

With VoiceOver (Mac / iOS) Mac:

- `VO + U` to open the rotor - select Headings to navigate by heading
- `VO + Command + F` to search the page
- `H` (with Quick Nav on) to move between headings

iOS:

- Swipe left/right to navigate elements
- Use the rotor (`two-finger rotate`) to set navigation mode to Headings
- Double-tap to activate links

Finding What You Need

If you know which unit or project you need Open the table of contents and look for the unit or project name. The structure follows this pattern:

- Unit 0 - Foundation lessons (safety, how printing works, calipers, OpenSCAD basics, slicing)
- Unit 1 - Guided projects (Project 0 and Project 1)
- Unit 2 - Intermediate skills (parametric design, tolerances, advanced slicing, materials)
- Unit 3 - Open-ended projects (Project 2, 3, and 4)
- Reference Materials - Quick-reference sheets you can keep open while working
- PowerShell Foundation - Command-line navigation guide

If you are looking for a specific term or command Use the Search function (`S`). Search for:

- An OpenSCAD command (e.g., `difference`, `translate`, `module`)
- A vocabulary word (e.g., `infill`, `tolerance`, `stakeholder`)
- A project name (e.g., `floor marker`, `jewelry`, `assistive technology`)

If you are looking for reference material while working Keep a second browser tab open to the Reference Materials section. Useful pages to bookmark:

- OpenSCAD Cheat Sheet
- Slicing Settings Quick Reference
- Filament Comparison Table
- Screen Reader Coding Tips (NVDA/JAWS)

Printing or Saving Pages

To save or print any page for offline use:

- `Ctrl + P` opens the print dialog in any browser
- Use ”Save as PDF” to save a local copy
- For the whole book: if your instructor has provided a PDF version, use that - it contains all chapters in one file

Reporting a Problem

If a page is missing content, has a broken link, or is difficult to navigate with your screen reader, let your instructor know:

- Which page (chapter title)
- What you were trying to do
- What happened instead

This helps improve the curriculum for future students.

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

Screen Reader Coding Tips - NVDA & JAWS

General Principles for Coding with a Screen Reader

- Turn punctuation up. All OpenSCAD syntax - semicolons, brackets, parentheses, commas - is meaningful. If your screen reader skips punctuation, you will miss syntax errors. Set punctuation to Most or All before coding.
- Navigate by line. Arrow up and down to move through code one line at a time. Use `Ctrl + Left/Right` to move word by word within a line.
- Use go-to-line. Both VSCode and Notepad++ let you jump to a specific line number with `Ctrl + G`. OpenSCAD errors always give a line number - use this to find them fast.
- Spell when uncertain. If you are not sure what a character is, spell the line character by character using your screen reader's spell command.
- Comment as you go. Adding a short comment after a block of code lets you navigate by searching for recognizable words with `Ctrl + F`.

NVDA Quick Reference

Setting Punctuation Level `NVDA + P` - cycles through: None -> Some -> Most -> All

For code: set to Most or All.

You can also set a permanent default: `NVDA + N` -> Preferences -> Settings -> Speech -> Symbol level

Essential Reading Commands

Action	Keys
Read current line	`NVDA + Up Arrow`
Spell current line	`NVDA + Up Arrow` (twice quickly)
Read from cursor to end	`NVDA + Down Arrow`
Read current word	`NVDA + Numpad 5`
Spell current word	`NVDA + Numpad 5` (twice quickly)
Read current character	`NVDA + Numpad 2`
Stop reading	`Ctrl`

Navigation in VSCode with NVDA

Action	Keys
Move by character	`Left / Right Arrow`
Move by word	`Ctrl + Left / Right Arrow`
Move by line	`Up / Down Arrow`
Start / end of line	`Home / End`
Start / end of file	`Ctrl + Home / Ctrl + End`
Go to line number	`Ctrl + G`, type number, `Enter`
Find text	`Ctrl + F`
Toggle line comment	`Ctrl + /`

If NVDA Stops Reading the Editor

1. Click in the editor area once with the mouse (or press `Escape` and then click)
2. Press `NVDA + Space` to toggle between Browse and Application mode - for code editors, you want Application mode

JAWS Quick Reference

Setting Punctuation Level `JAWS Key + Shift + 2` - cycles through punctuation levels

For code: set to All.

Essential Reading Commands

Action	Keys
Read current line	`JAWS Key + Up Arrow`
Spell current line	`JAWS Key + Up Arrow` (twice quickly)
Read from cursor	`JAWS Key + A`
Read current word	`JAWS Key + Numpad 5`
Read next word	`JAWS Key + Right Arrow`
Read current character	`JAWS Key + Numpad 5` (once = word, twice = spell)
Stop reading	`Ctrl`

Adjusting Speech Rate

Action	Keys
Increase rate	`Alt + Ctrl + Page Up` (or `Fn + Up Arrow` on laptops)
Decrease rate	`Alt + Ctrl + Page Down` (or `Fn + Down Arrow` on laptops)

Navigation in VSCode with JAWS

Action	Keys
Move by character	`Left / Right Arrow`
Move by word	`Ctrl + Left / Right Arrow`
Move by line	`Up / Down Arrow`
Start / end of line	`Home / End`
Start / end of file	`Ctrl + Home / Ctrl + End`
Go to line number	`Ctrl + G`
Find text	`Ctrl + F`
Toggle line comment	`Ctrl + /`

If JAWS Stops Reading the Editor

1. Press `JAWS Key + Z` to toggle Virtual/Browse mode off - for VSCode you want Virtual mode off
2. Focus the menu bar with `Alt`, then press `Escape` to return to the editor
3. If still not working, press `Alt + F4` to close VSCode and reopen it

Window Focus When you launch OpenSCAD from VSCode (via the task runner), focus stays in VSCode. To switch to OpenSCAD to read the console or error messages:

- `Alt + Tab` to cycle through open windows
- JAWS will announce the application name as you cycle

OpenSCAD-Specific Tips

Reading Errors When OpenSCAD can't render your code, it outputs an error. The error message always includes:

- Line number - use `Ctrl + G` in VSCode to jump there
- Error type - usually a missing semicolon, a typo in a command name, or mismatched brackets

Common errors and what they sound like:

Error message	What it usually means
”Expected ';' ...”	You forgot a semicolon at the end of a statement
”Expected ',' or ')' ...”	Missing comma between parameters, or unclosed parenthesis
”Identifier ... is undefined”	You typed a variable name wrong, or used a variable before declaring it
”WARNING: Normalized tree is empty”	Your shape has no geometry (e.g., you subtracted more than you started with)

Bracket Matching All OpenSCAD shapes and operations use brackets and braces:

- Parentheses `()` - hold parameters: `cube([10, 10, 10])`
- Square brackets `[]` - hold vectors (lists of numbers): `[10, 10, 10]`
- Curly braces `{}` - hold groups of shapes for boolean operations

Every opening bracket must have a closing bracket. If you are missing one, OpenSCAD will report an error somewhere near (but not always exactly at) the problem.

VSCode can help: when your cursor is on a bracket, VSCode highlights the matching bracket. With NVDA, you can navigate to the matching bracket with `Ctrl + Shift + \`.

Commenting Out Code To test part of your code without deleting it, comment it out:

- Single line: position cursor at start of line, type `//`
- Multiple lines: select the lines, press `Ctrl + /` in VSCode (adds `//` to each selected line)
- Block comment: type `/*` before and `*/` after the block

To uncomment: select the commented lines, press `Ctrl + /` again.

Navigating Large Files Use `Ctrl + F` (Find) to locate sections of your code:

- Search for your module names to jump to them: e.g., `module round_bead`

- Search for comments you wrote: e.g., `// Step 2`
- Search for line numbers in error messages

Caliper and OpenSCAD Workflow Tips

When measuring an object and entering it into OpenSCAD, say the measurement aloud before typing it to reduce transcription errors. Then read the number back after typing it to confirm.

Recommended sequence:

1. Measure -> say ”seventy point three millimeters”
2. Type `70.3` in OpenSCAD
3. Read back: ”seven zero point three” to confirm

mdBook Navigation (Web Version of This Curriculum)

If you are reading this curriculum through the web version (mdBook), here are tips for navigating with a screen reader:

NVDA + Browser

- `H` to jump between headings (chapter navigation)
- `Ctrl + F` to search within the current page
- The sidebar table of contents is a navigation landmark - use `NVDA + F7` to list landmarks, or press `D` to jump between landmark regions
- Previous/Next chapter links are at the bottom of each page

JAWS + Browser

- `H` to move between headings
- `Ctrl + F` (browser Find) to search
- `JAWS Key + F6` to list headings
- `R` to jump to regions/landmarks
- Use the search box in the mdBook header to search across all chapters

Keyboard Navigation (No Screen Reader)

- `Left Arrow / Right Arrow` - previous/next chapter
- `S` - focus the search box
- `Escape` - close search results
- `T` - toggle the table of contents sidebar

References

NV Access. (2024). *NVDA user guide*. https://www.nvaccess.org/files/nvda/documentation/userGuide.html

Freedom Scientific. (2024). *JAWS for Windows help*. https://support.freedomscientific.com/Content/Documents/Manuals/JAWS/JAWS-Screen-Reader-Help.pdf

Microsoft. (2024). *Visual Studio Code accessibility*. https://code.visualstudio.com/docs/editor/accessibility

More on mdBook navigation

- mdBook (general / keyboard navigation): https://rust-lang.github.io/mdBook/ - includes documentation and basic navigation/usage for the mdBook web UI.
- Curriculum mdBook navigation (with accessibility tips): mdBook Navigation Guide - local guide in this curriculum with notes for using mdBook with and without screen readers.

Other Screen Readers

Dolphin SuperNova (commercial) and Windows Narrator (built-in) are also supported; the workflows and recommendations in this document apply to them. See https://example.com and https://example.com for vendor documentation.

Your First Print - Guided Extension

Estimated time: 2-4 hours (including setup and print monitoring)

Learning Objectives

- Select a simple ready-made model and evaluate its printability for a classroom printer
- Configure slicer settings for a short-duration print and prepare the printer safely
- Document print parameters and reflect on the physical outcome

Materials

- Computer with slicer and access to the online repository
- Prusa Mini+ or classroom-approved printer, filament spool

Step-by-step Tasks

1. Choose a simple model (<2 hours print) from Thingiverse or Printables and save the STL.
2. Inspect the model: note overhangs, thin features, and dimensions; write two short reasons why this model is appropriate for a first print.

3. Load the model in your slicer, select the classroom profile, and adjust settings only if necessary (layer height, infill, supports). Record the final print time and filament estimate.
4. Perform safety checks, start the print, and monitor the first 10 minutes for adhesion and extrusion problems.
5. After cooling, measure three critical dimensions and compare to the models stated dimensions; record deviations.

Starter Code

You can use this basic project scaffold as your starting point:

```
// Basic Project Scaffold for 3D Printing
// This template provides a starting point for beginner projects
// Complete the TODO sections to create your own design
// ============================================
// PROJECT CONFIGURATION
// ============================================
// TODO: Set your project name
projectname = "My First Project";
// TODO: Define overall dimensions (mm)
objectwidth = 50;
objectdepth = 50;
objectheight = 20;
// Wall thickness for strength (2-3mm recommended)
wallthickness = 2;
// ============================================
// MATERIAL PROPERTIES
// ============================================
// Print resolution (0.2mm layers recommended)
$fn = 30; // Fragment quality (higher = smoother curves, slower rendering)
// ============================================
// MAIN DESIGN
// ============================================
module baseshape() {
    // TODO: Replace this cube with your design
    cube([objectwidth, objectdepth, objectheight], center = true);
}
module hollowversion() {
    // Creates a hollow version by subtracting an inner cube
    difference() {
        baseshape();
        // Inner void (adjust padding as needed)
        translate([0, 0, 0.5])
            cube([objectwidth - 2*wallthickness,
                 objectdepth - 2*wallthickness,
                 objectheight - 2*wallthickness],
                center = true);
    }
}
// Render the solid version for your first print
baseshape();
```

Probing Questions

- Why did you select this model? What risks did you anticipate and how did you mitigate them?
- Which slicer setting most affects print time for this model and why?
- If a thin feature failed, what minimal change would you make to ensure success next time?

Quiz - Your First Print (10 questions)

1. What is the first-check you do after loading filament? (short answer)
2. Name two slicer settings that affect strength. (short answer)
3. Why monitor the first layers of a print? (one sentence)
4. How do you document filament used for reproducibility? (short answer)
5. What is one sign of poor bed adhesion? (one sentence)
6. True/False: Selecting a model with support requirements is not recommended for your first print. (Answer: True)
7. Short answer: Describe two characteristics of a ”printability-friendly” model that would be good for a beginner's first print.
8. Practical scenario: Your first print is showing stringing (thin lines between parts). What are two possible causes and how would you troubleshoot?
9. Multiple choice: When measuring a printed dimension, where should you measure multiple times (three different spots) to account for print variation? (A) Never, measure once (B) Only if you suspect error (C) Always measure at least three locations - Answer: C
10. Reflection: Explain why carefully selecting your first print model (simple, robust, known to work on your printer) is better than immediately attempting a complex design. What will you learn from success that prepares you for harder projects?

Extension Problems (10)

1. Re-slice the model with a finer layer height and compare surface finish and print time; document differences.
2. Modify the model in OpenSCAD to thicken a failing feature and reprint a small test piece.
3. Create a short checklist script (or text checklist) that verifies spool metadata and bed temperature before printing.
4. Produce a one-page reflection that includes three lessons learned and one parameter you will change next time.
5. Share your measurements and photos in the class folder and give feedback on two peers' prints.
6. Conduct a post-print analysis: compare actual measurements to STL specifications; identify and document any deviations.
7. Print the same model in two different materials or with two different slicer profiles; compare durability, appearance, and accuracy.
8. Create a detailed documentation package: CAD file, slicer settings, print log, measurements, and lessons learned.
9. Design a quality assurance test: define pass/fail criteria and systematically verify your print meets all requirements.
10. Write a ”first print troubleshooting guide” based on your experience: common issues you encountered and how you solved them.

Deliverables

- Short report: model chosen, key slicer settings, measured deviations, and answers to probing questions.
- Photos of the final print and the measured values table.

Your First Print - Student Documentation Template (Extension Project)

- Author:
- Date:
- Description: Select a simple 3D model, configure slicer settings, and complete your first independent print job.

Ideas and Concept

Model Selection

- Model name and source (Thingiverse, Printables, etc.):
- Link or file reference:
- Why did you choose this model?

Printability Assessment

- Estimated print time (from slicer):
- Estimated filament:
- Expected challenges (overhangs, thin features, supports needed):
- Why is this model appropriate for a first print?

Measurements and Printer Configuration

Printer Setup Log

Parameter	Value
Printer model	
Nozzle diameter	
Bed temperature	
Nozzle temperature	
Layer height	
Infill percentage	
Support settings	

Safety Checks (Completed)

☐ Bed clean and level

☐ Bed adhesive/prep applied (if needed)
☐ Filament loaded correctly
☐ Nozzle at correct height
☐ Print area clear of obstructions
☐ Buildplate secured

Object Notes

Print Monitoring Notes

- What did you observe in the first 10 minutes?
- Any adhesion issues?
- Any extrusion problems?
- Print completed successfully: Yes / No

Printed Part Measurements (After Cooling)

Feature	Model Spec (mm)	Printed (mm)	Deviation	Notes

Use and Assembly Notes

- How does the print feel and look compared to the model?
- Did any features fail or print poorly?
- If there were issues, what do you suspect caused them?

Reflections

What Went Well

- Which aspect of the print was most successful?
- Did anything surprise you positively?

What Didn't Go Well

- Were there any defects or failed features?
- What challenges did you encounter?

Learning and Next Steps

- What did you learn from this first independent print?
- What would you do differently next time?
- If you printed this again, what settings would you change?

Post-Print Analysis

- Compare this print to others you've seen in the classroom
- What variables (temperature, layer height, infill) do you think most affected the outcome?
- How would you test that hypothesis?

Attachments

- ☐ Exported `.stl` model file (or link to source)
- ☐ Slicer settings (screenshot or export)
- ☐ Photo of final print (multiple angles if possible)
- ☐ Safety checklist (signed/dated)
- ☐ Print log with timestamps (if available from printer)

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Feedback:

Resubmission (if applicable)

What Was Changed (One-paragraph explanation of changes made and why)

Revised Score

Category	Revised Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Your First Print - Teacher Template (Extension Project)

Briefing

A foundational extension project where students select a ready-made 3D model from online repositories, configure slicer settings, and complete their first independent print job. This project builds confidence and establishes safe printing practices.

Key Learning: Printer safety; slicer configuration; print monitoring; dimensional verification.

Real-world Connection: Production environments rely on safe print setup and quality verification. This project establishes habits that transfer to any 3D printing context.

Constraints

- Model must be 2 hours print time
- No or minimal support requirements recommended
- Student must document printer configuration and safety checks
- Student must compare printed dimensions to model specifications

Functional Requirements

- Model selected and deemed appropriate for beginner
- Printer configured safely with classroom profile
- Print completes successfully with good adhesion
- Printed part is measured and deviations documented

Deliverables

- Completed documentation template with:
  - Model selection rationale and printability assessment
  - Printer setup log (materials, temperature, settings)
  - Print monitoring observations (first 10 minutes, monitoring during print)
  - Dimensional measurements and deviation analysis
  - Reflection on first print experience
  - Post-print analysis and troubleshooting notes
- Photograph of final print
- Slicer settings (screenshot or text export)

Rubric

All projects are scored on a 0-9 scale across three equally weighted categories (3 points each):

Category	Points	What We Measure
Problem & Solution	0-3	Did the student select an appropriate model? Did the print succeed?
Design & Code Quality	0-3	Is documentation of printer setup thorough? Is print quality good?
Documentation	0-3	Is all documentation complete? Are measurements recorded? Is reflection thoughtful?

Category 1: Problem & Solution (0-3 points)

Score	Description
3	Student selected appropriate model with sound reasoning. Print completed successfully with good adhesion and finish. No major issues.
2	Model generally appropriate. Print mostly successful with minor adhesion or finish issues.
1	Model had some difficulties; print partially successful or required intervention.
0	Print failed or model was inappropriate.

Category 2: Design & Code Quality (0-3 points)

Score	Description
3	Printer setup documented thoroughly. Safety checks evident. Print quality excellent. Photos provided.
2	Setup documented adequately. Print quality acceptable. Some documentation of settings.
1	Minimal documentation of setup. Print quality has defects.
0	Setup not documented or print failed to start.

Category 3: Documentation (0-3 points)

Score	Description
3	All sections complete. Measurements recorded precisely. Reflection is specific and shows learning.
2	Most sections present. Measurements recorded but reflection brief.
1	Incomplete sections. Measurements minimal. Reflection lacking.
0	No documentation.

Score Interpretation

Total Score	Interpretation	Next Step
8-9	Excellent work	Student ready for independent printing projects
6-7	Good execution	Encourage documentation habit
4-5	Meets basics	Discuss printer setup and safety
2-3	Does not meet expectations	Resubmission or additional practice

Total Score	Interpretation	Next Step
0-1	Missing major components	Meet with instructor

Resubmission Policy

Students may resubmit to improve their score. Resubmissions must include:

1. A one-paragraph explanation of what was changed and why

The resubmission score replaces the original.

Assessment Notes

- Strong submissions show careful model selection reasoning, thorough printer documentation, and specific observations about print behavior
- Watch for: Poor printer documentation, dismissive attitude toward safety checks, or generic reflections
- Reinforce: Safety first; documentation enables learning and troubleshooting
- Next Step: Extension problem suggestions include print quality analysis or variant material testing

Lesson 2: Geometric Primitives and Constructive Solid Geometry

Estimated time: 75-90 minutes

- Use OpenSCAD primitives (`cube()`, `sphere()`, `cylinder()`) and transforms (`translate()`, `rotate()`, `scale()`)[58]
- Apply CSG operators (`union`, `difference`, `intersection`) safely and diagnose common numerical issues[59]
- Use quick diagnostic renders and validate geometry in a slicer[60]

Materials

- 3dMake project scaffold with `src/main.scad`
- Example primitive snippets (provided in assets)

[58] OpenSCAD Manual - Primitives and Transforms - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Using_the_2D_Subsystem

[59] Gonzalez Avila, J. F., Pietrzak, T., & Casiez, G. (2024). *Understanding the challenges of OpenSCAD users for 3D printing*. Proceedings of the ACM Symposium on User Interface Software and Technology.

[60] Slicer Validation - PrusaSlicer Documentation - https://docs.prusa3d.com/en/guide/39012-validation-tools/

- Reference: openscad-cheat-sheet.md for syntax quick-reference

1. Open `src/main.scad`; identify and run simple examples using `cube()`, `sphere()`, and `cylinder()` [61].

   Example primitives:

```
// Primitive shapes - uncomment one to try
// Cube (x, y, z dimensions)
cube([20, 20, 20]);
// Sphere (radius, resolution)
// sphere(r=10, $fn=32);
// Cylinder (radius, height, resolution)
// cylinder(r=10, h=20, $fn=32);
```

2. Create three short examples demonstrating `union()`, `difference()`, and `intersection()` and render with reduced `$fn`. Review CSG best practices[62].

   Example CSG operations:

```
// UNION - Combine multiple shapes
union(){
  cube([20, 20, 20]);
  translate([15, 0, 0]) sphere(r=10, $fn=32);
}
// DIFFERENCE - Subtract shapes
// difference(){
// cube([20, 20, 20]);
// translate([10, 10, 10]) sphere(r=8, $fn=32);
// }
// INTERSECTION - Keep only overlapping volumes
// intersection(){
// cube([20, 20, 20]);
// sphere(r=12, $fn=32);
// }
```

3. Reproduce a failing `difference()` case and apply the 0.001 offset strategy to the subtractor; re-render and confirm fix. This technique addresses common manifold issues[63].

   Fix technique:

```
// Problem: Coincident faces cause non-manifold issues
difference(){
  cube([20, 20, 20], center=true);
  // Use translate with small offset (0.001) to prevent coincident →
     ↪ faces
  translate([0, 0, 0.001]) sphere(r=10, $fn=32);
}
```

[61]OpenSCAD Manual - Primitives and Transforms - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Using_the_2D_Subsystem

[62]Gonzalez Avila, J. F., Pietrzak, T., & Casiez, G. (2024). *Understanding the challenges of OpenSCAD users for 3D printing*. Proceedings of the ACM Symposium on User Interface Software and Technology.

[63]Google. (2025). *Vertex AI Gemini 3 Pro Preview: Getting started with generative AI*. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3

4. Build an STL with `3dm build` and open it in your slicer to check for thin walls or islands[64].

5. Document any fixes in the project README and commit the working `main.scad` and STL.

Modifier Characters for Advanced Debugging

OpenSCAD includes special modifier characters that help you isolate and debug specific parts of your model. These single-character prefixes (placed before module/shape names) change how that object is rendered without affecting the underlying code.

The Four Modifier Characters

1. The `!` Character (Show Only) Use `!` to render only the marked object and hide everything else. This is essential for isolating geometry when debugging complex models:

```
module base() {
  cube([50, 50, 5]);
}
module stand() {
  translate([0, 0, 5])
    rotate([60, 0, 0])
    cube([50, 5, 40]);
}
module lip() {
  translate([0, 50, 45])
    cube([50, 8, 15]);
}
// Show all parts
base();
stand();
lip();
// DEBUG: Show only the stand by adding !
// !stand();
```

When to use: You're debugging a multi-part assembly and want to see one component in isolation

2. The # Character (Highlight) Use # to render the marked object and highlight it in red (in preview mode). This helps you see which part corresponds to which code:

```
difference() {
  cube([30, 30, 30], center=true);
  // Highlight the subtracted sphere in red
  #sphere(r=15, $fn=32);
}
```

[64]Slicer Validation - PrusaSlicer Documentation - https://docs.prusa3d.com/en/guide/39012-validation-tools/

When to use: Identifying which boolean operation is creating problems or validating that two objects overlap correctly

3. The % Character (Transparent/Ghost) Use % to render the marked object as transparent/ghosted (shown as semi-transparent geometry). This helps you see what's ”underneath” without removing the code:

```
module outer_shell() {
  difference() {
    cube([50, 50, 50], center=true);
    cube([44, 44, 44], center=true); // Wall thickness of 3mm
  }
}
module internal_support() {
  translate([0, 0, -10])
    cube([20, 20, 10], center=true);
}
// Show shell normally and support as ghosted
outer_shell();
%internal_support(); // Transparent - see where it sits relative to shell
```

When to use: Visualizing assembly relationships or checking alignment of internal vs external features

4. The * Character (Disable) Use * to disable rendering of a marked object entirely. It's like commenting it out but without modifying the code:

```
module component_a() { cube([30, 30, 30]); }
module component_b() { sphere(r=20, $fn=32); }
module component_c() { cylinder(h=40, r=15, $fn=32); }
component_a();
*component_b(); // Disabled - won't render
component_c();
```

When to use: Testing whether a component is causing rendering errors or print failures without deleting code

Practical Debugging Workflow Using Modifiers

Scenario: Your phone stand model has three parts, but it's rendering very slowly. You suspect one part is the problem.

```
module base() {
  cube([100, 100, 5]);
}
module stand() {
  // Complex geometry with many $fn values
  translate([0, 0, 5])
    rotate([60, 0, 0])
    minkowski() {
      cube([90, 10, 40], center=true);
      cylinder(r=5, h=0.01, $fn=64); // This might be slow!
    }
}
module lip() {
  translate([0, 100, 45])
    cube([100, 10, 15]);
}
```

```
// DEBUGGING STEPS:
// Step 1: Show all parts (original speed problem)
base();
stand();
lip();
// Step 2: Disable stand to test if it's the culprit
// base();
// *stand();
// lip();
// Result: If still slow, problem is elsewhere. If fast, stand() is the issue
    .
// Step 3: If stand() is slow, highlight just that part
// !#stand(); // Shows only stand in red - can now debug its geometry
// Step 4: Reduce resolution in stand() and test again
// (modify the $fn=64 to $fn=16 and see if faster)
```

Key Points About Modifiers

Character	Effect	Use Case
!	Show only this object	Isolate one component in multi-part assembly
#	Highlight in red	See what's being subtracted/added in boolean ops
%	Ghost/transparent	See assembly relationships and overlaps
*	Disable/hide	Test if a component is causing problems

Important

Modifiers only affect the visual preview/render. They do not change the exported STL file. When you export, all objects are included normally (unless you've actually deleted or commented them out).

Practice Exercise: Modifier Workflow

Create a file with three overlapping shapes and practice:

1. Use ! to show only the middle shape
2. Use # to highlight the bottom shape
3. Use % to ghost the top shape
4. Use * to disable all shapes one at a time

```
// Practice: Multi-part assembly debugging
module bottom_part() {
  cube([50, 50, 10], center=true);
}
module middle_part() {
  translate([0, 0, 15])
    cylinder(h=20, r=20, $fn=32);
}
module top_part() {
  translate([0, 0, 40])
    sphere(r=15, $fn=32);
}
```

```
// Test 1: Show only middle
// !middle_part();
// Test 2: Highlight bottom
// #bottom_part();
// middle_part();
// top_part();
// Test 3: Ghost top
// bottom_part();
// middle_part();
// %top_part();
// Test 4: Disable one at a time
// *bottom_part();
// middle_part();
// top_part();
```

Debugging, Resolution, and Common Issues

As your OpenSCAD models become more complex, you'll encounter rendering errors, non-manifold geometry, or slow previews. Learning to debug efficiently is essential for productive design work.

Understanding OpenSCAD Console Errors When you press F5 to preview or F6 to render your model, OpenSCAD displays warnings and errors in the console at the bottom of the editor. Here's what to look for:

- Error Line Numbers: The console message tells you the exact line where the problem was detected. Look for missing semicolons, unmatched braces, or undefined variables
- Syntax Errors: These prevent the model from rendering at all (common causes: missing `;`, unclosed `{` or `[`, typos in function names)
- Warnings (Non-Manifold Geometry): Your geometry is *technically* valid but has internal inconsistencies that may prevent printing

Example Console Output:

```
ERROR: Unexpected token 'sphere' at line 15, column 5
  Expected one of: } ) ]
```

This tells you that line 15 has a syntax issue-likely a missing closing bracket on the previous line.

The `0.001` Offset Strategy (Revisited) One of the most common issues with CSG operations is coincident faces-when two shapes touch exactly at their boundaries. This creates a non-manifold geometry that slicer tools flag as unprintable.

The Fix: Add a tiny offset (`0.001` mm or smaller) to move one surface slightly:

```
// BEFORE (problematic):
difference(){
  cube([20, 20, 20], center=true);
  sphere(r=10, $fn=32);
}
// AFTER (fixed):
```

```
difference(){
  cube([20, 20, 20], center=true);
  translate([0, 0, 0.001]) sphere(r=10, $fn=32); // Tiny offset prevents
      coincident faces
}
```

This 0.001 mm offset is invisible to the human eye but fixes the non-manifold warning and makes the model printable.

Using `$fn` to Balance Speed and Quality The `$fn` parameter controls the resolution of curved surfaces (spheres, cylinders, etc.). It represents the number of polygonal faces used to approximate the curve:

- `$fn = 12` - Very fast preview, coarse geometry (useful for quick testing)
- `$fn = 32` - Good balance for most designs (default for many situations)
- `$fn = 100+` - High quality but slow to render; use only for final export

Debugging Workflow:

1. During design iteration: Use `$fn = 12` or `$fn = 20` for quick feedback
2. Before export: Increase to `$fn = 32` or higher for final quality
3. For complex assemblies: Keep `$fn` low during iteration to save preview time

Incremental Rendering: The F5/F6 Workflow

- F5 (Preview): Quick render to check overall geometry and catch layout errors
- F6 (Full Render): Slower but more accurate; use before export
- Strategy: Press F5 frequently while editing to get immediate feedback, then F6 once before exporting

This workflow catches errors early and saves you from spending 10 minutes rendering only to discover a syntax error.

Diagnosing Non-Manifold Geometry A model might render in OpenSCAD but still be unprintable. The slicer will report errors like "non-manifold edges" or "holes in surface." Common causes:

1. Coincident Faces -> Use small offsets (0.001 mm)
2. Zero-Thickness Walls -> Ensure all walls are at least 0.8-1.0 mm
3. Gaps Between Shapes -> Check that boolean operations have no small gaps; use `minkowski()` for rounded edges if needed
4. Inside-Out Geometry -> A shape might be flipped; use `scale([1, 1, -1])` to flip normals if needed

When your slicer reports non-manifold issues, use the slicer's visualization to locate the problem, then reference the line numbers in your OpenSCAD code and test fixes incrementally using F5/F6.

Checkpoints

- After task 3 the problematic boolean should render without non-manifold warnings[65].
- After this section, you should be able to interpret at least three common console error messages and apply appropriate fixes.

Quiz - Lesson 3dMake.2 (10 questions)

1. Name three primitive functions in OpenSCAD[66].
2. What does `difference()` accomplish[67]?
3. Why might two coincident faces cause a render failure[68]?
4. What is the 0.001 rule and why is it useful[69]?
5. How does lowering `$fn` help during debugging[70]?
6. True or False: `union()` combines shapes while `intersection()` keeps only overlapping volumes.
7. Describe what Constructive Solid Geometry (CSG) is and give an example of where it's useful.
8. Explain how scale(), translate(), and rotate() transforms change a primitive's behavior.
9. How would you diagnose and fix a non-manifold issue in your model?
10. When validating geometry in a slicer, what specific issues should you look for?

Lesson 2 Assets & Projects

Your Lesson 2 assets are located in the centralized assets folder:

- Lesson 2 Asset Overview (assets/Lessons_3dMake_2/README.md) - Folder contents and learning resources
- Your Second Print Project (assets/Lessons_3dMake_2/Your_Second_Print/) - Adapt models to real-world needs
- Bonus Print Project (assets/Lessons_3dMake_2/Bonus_Print/) - Practice resizing and parametric variation

[65] Google. (2025). *Vertex AI Gemini 3 Pro Preview: Getting started with generative AI.* https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3

[66] OpenSCAD Manual - Primitives and Transforms - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Using_the_2D_Subsystem

[67] Gonzalez Avila, J. F., Pietrzak, T., & Casiez, G. (2024). *Understanding the challenges of OpenSCAD users for 3D printing.* Proceedings of the ACM Symposium on User Interface Software and Technology.

[68] Google. (2025). *Vertex AI Gemini 3 Pro Preview: Getting started with generative AI.* https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3

[69] Google. (2025). *Vertex AI Gemini 3 Pro Preview: Getting started with generative AI.* https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3

[70] Gonzalez Avila, J. F., Pietrzak, T., & Casiez, G. (2024). *Understanding the challenges of OpenSCAD users for 3D printing.* Proceedings of the ACM Symposium on User Interface Software and Technology.

These projects reinforce the boolean operations and CSG concepts you learned in this lesson.

Advanced Example: Rotational and Linear Extrusion

Beyond basic primitives and boolean operations, OpenSCAD provides powerful 2D-to-3D operations. Here are practical examples from real-world designs:

Example 1: Rotating 2D Shapes with `rotate_extrude()`

The `rotate_extrude()` function spins a 2D profile around the Z-axis, creating axially symmetric objects like vases, bowls, or cones:

```
// Simple Vase - Rotating a 2D profile
$fn = 100;
rotate_extrude(angle = 360, convexity = 4)
{
  translate([20, 0, 0])
  square([8, 50], center=false);
}
```

What it does:

- Draws a 2D square 20 units from the center
- Rotates it 360deg around the Z-axis
- Creates a hollow cylindrical vase

Try this variation:

```
// Vase with a curved profile
$fn = 100;
rotate_extrude(angle = 360, convexity = 4)
{
  // Profile shape: wider at top, narrower at bottom
  polygon([
    [20, 0], // bottom inner
    [25, 0], // bottom outer
    [30, 40], // top outer
    [25, 40] // top inner
  ]);
}
```

Example 2: Linear Extrusion with 2D Text

The `linear_extrude()` function pulls a 2D profile vertically (along the Z-axis). Combined with the `text()` function, it creates embossed letters:

```
// Embossed Text - Name Plate
linear_extrude(height = 5, convexity = 10)
{
  text("MAKER", size = 30, font = "Impact:style=Regular",
      halign = "center", valign = "center");
}
```

What it does:

- Creates 2D text outline
- Pulls it 5 mm upward
- Creates a 3D embossed nameplate

Complete nameplate with backing:

```
// Name plate with base and embossed text
union()
{
  // Base plate
  cube([120, 40, 3], center = true);
  // Embossed text
  translate([0, 0, 2])
  linear_extrude(height = 2)
  {
    text("MAKER", size = 24, font = "Impact:style=Regular",
         halign = "center", valign = "center");
  }
}
```

Example 3: Practical Project - Hook with `rotate_extrude()`

This real-world example creates a wall mount hook by rotating a 2D profile:

```
// Wall Mount Hook
$fn = 100;
difference()
{
  // Outer hook profile - rotated 360deg
  rotate_extrude(angle = 360, convexity = 4)
  {
    translate([25, 0, 0])
    square([8, 60], center = false);
  }
  // Hollow center
  translate([0, 0, -5])
  cylinder(h = 70, d = 40);
}
```

Key observations:

- The 2D square is positioned 25 units from the Z-axis
- Rotating it creates a thick-walled hollow cylinder
- This is used in real products like the birdhouse and PVC hook examples from the *Simplifying 3D Printing* textbook

Key Concepts for 2D-to-3D Conversion

Function	Purpose	Common Use
`linear_extrude(height)`	Pull 2D shape vertically	Text, badges, plaques
`rotate_extrude(angle)`	Rotate 2D shape around Z-axis	Vases, bowls, hooks, symmetric parts
`scale()`	Stretch/compress along axes	Tapered cones, proportional variations

Function	Purpose	Common Use
Combining with `difference()`	Hollow out the extruded shape	Cups, containers, hollow handles

Checkpoint: After this section, you can:

- Use `linear_extrude()` to create embossed text and flat designs
- Use `rotate_extrude()` to create axially symmetric 3D shapes
- Combine extrusion with boolean operations to hollow or refine shapes

Extension Problems (10)

1. Create a small assembly using `union()` of three primitives and export the STL. Reference best practices from OpenSCAD documentation[71].
2. Intentionally create a failing boolean and fix it using offsets; explain your approach. Document the manifold issues encountered[72].
3. Write a short test script that generates three variants with varying `$fn` values and compare render times. Consider using 3dMake workflows[73].
4. Use `3dm info` (if available) to generate a report on your model and document any recommendations[74].
5. Explore using a library module (e.g., a fillet helper) to fix a sharp corner and note the difference in final STL[75].
6. Build a geometry validation toolkit: test all basic transformations (union, difference, intersection) with edge cases and document failure modes.
7. Create a parametric assembly generator that produces 5+ configurations and validates each for printability.
8. Develop a boolean operation troubleshooting guide with visual examples and step-by-step fixes.
9. Design a library module system for common geometric operations (fillets, chamfers, arrays); test reusability across projects.
10. Write an accessibility guide for boolean operations: describe methods for validating geometry non-visually using model properties and slicer feedback.

[71]OpenSCAD Manual - Primitives and Transforms - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Using_the_2D_Subsystem

[72]Google. (2025). *Vertex AI Gemini 3 Pro Preview: Getting started with generative AI*. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3

[73]Slicer Validation - PrusaSlicer Documentation - https://docs.prusa3d.com/en/guide/39012-validation-tools/

[74]Slicer Validation - PrusaSlicer Documentation - https://docs.prusa3d.com/en/guide/39012-validation-tools/

[75]Gonzalez Avila, J. F., Pietrzak, T., & Casiez, G. (2024). *Understanding the challenges of OpenSCAD users for 3D printing*. Proceedings of the ACM Symposium on User Interface Software and Technology.

Supplemental Resources

For additional examples and practice with 3D primitives and boolean operations, explore these resources:

- Programming with OpenSCAD EPUB Textbook - Comprehensive guide to primitives, boolean operations, and CSG fundamentals
- CodeSolutions: 3D Primitives - Working examples of cube, sphere, cylinder, and other 3D shapes
- CodeSolutions: Combining Shapes - Examples of union, difference, intersection, and hull operations
- Practice Worksheets: 3D Shapes - Visualization and decomposition exercises

OpenSCAD Quick Reference - Cheat Sheet

Keep this handy during all OpenSCAD work. For full documentation: https://openscad.org/documentation.html

Basic Shapes (Primitives)

```
cube([length, width, height]); // rectangular box
cube([l, w, h], center = true); // centered at origin

sphere(r = radius); // sphere by radius
sphere(d = diameter); // sphere by diameter

cylinder(h = height, r = radius); // cylinder
cylinder(h = height, d = diameter); // cylinder by diameter
cylinder(h = height, r1 = 5, r2 = 2); // cone (different top/bottom radii)

$fn = 50; // sets smoothness for spheres/cylinders (higher = smoother, slower →
    ↪ )
```

Transformations

```
translate([x, y, z]) shape; // move
rotate([x_deg, y_deg, z_deg]) shape; // rotate
scale([x, y, z]) shape; // scale (1.5 = 150%)
mirror([1, 0, 0]) shape; // mirror across YZ plane (use [0,1,0] for XZ,  →
    ↪ [0,0,1] for XY)
```

Boolean Operations

```
union() { shape1; shape2; } // combine shapes
difference() { base; subtract; } // remove one shape from another
intersection() { shape1; shape2; } // keep only overlapping region
```

Tip for difference(): Always make the subtracting shape 1 mm taller on both ends than the base shape to avoid zero-thickness artifacts.

```
// Example - box with a hole:
```

```
difference() {
    cube([30, 30, 10]);
    translate([15, 15, -1]) cylinder(h = 12, r = 5); // extends -1 to +11
}
```

Variables

```
length = 70; // define a variable
width = 16;
height = 5;

cube([length, width, height]); // use the variables
cube([length * 2, width, height]); // math works too
```

Modules (Reusable Functions)

```
// Define a module:
module my_box(l = 20, w = 15, h = 10) {
    cube([l, w, h]);
}

// Call the module:
my_box(); // uses all defaults
my_box(30, 20, 8); // positional arguments
my_box(l = 50); // named argument; others use defaults
```

Loops

```
// Repeat a shape N times:
for (i = [0 : 4]) { // i goes 0, 1, 2, 3, 4
    translate([i * 20, 0, 0]) cube([15, 15, 5]);
}

// Step size:
for (i = [0 : 5 : 20]) { // i goes 0, 5, 10, 15, 20
    translate([i, 0, 0]) sphere(r = 3);
}
```

2D Shapes (for extrusion)

```
circle(r = 10); // 2D circle
square([w, h]); // 2D rectangle
polygon([[0,0],[10,0],[5,10]]); // arbitrary 2D shape

// Extrude a 2D shape into 3D:
linear_extrude(height = 5) circle(r = 10); // makes a cylinder
rotate_extrude() translate([15, 0]) circle(r = 3); // makes a torus
```

Useful Functions

```
len([a, b, c]) // returns 3 (length of a vector)
sqrt(25) // returns 5
pow(2, 8) // returns 256 (2^8)
abs(-5) // returns 5
min(3, 5, 1) // returns 1
max(3, 5, 1) // returns 5
```

Comments

```
// Single-line comment

/*
  Multi-line
  comment
*/
```

Keyboard Shortcuts

Key	Action
F5	Preview (fast)
F6	Full render (for export)
Ctrl+S	Save
Ctrl+Z	Undo
F3	Reset camera view
F5 then scroll	Zoom with mouse

Export Workflow

1. Press F6 (full render - wait for it to complete)
2. File > Export > Export as STL
3. Save with a descriptive filename: `projectname_v2.stl`

Sources

OpenSCAD. (n.d.). *OpenSCAD cheatsheet*. https://openscad.org/cheatsheet/
OpenSCAD. (n.d.). *OpenSCAD documentation*. https://openscad.org/documentation.html
Gohde, J., & Kintel, M. (2021). *Programming with OpenSCAD*. No Starch Press. https://nostarch.com/programmingopenscad

Real-Life Problem Solver - Guided Extension

Estimated time: 4-8 hours (design, adapt, print, and document)

Learning Objectives

- Identify a user problem and evaluate candidate printed solutions
- Adapt an existing model to meet real constraints and safety requirements
- Test, measure, and iterate on a prototype with documented decisions

Materials

- Computer with slicer and access to repositories
- Printer, filament, basic hand tools for post-processing

Step-by-step Tasks

1. Interview a potential user (or yourself) and write a 1-paragraph problem statement.
2. Search repositories for candidate models; list three options and justify which one you will adapt.
3. Adapt the model (scale, add mounts, or modify features) and document the changes in a short changelog.
4. Slice, print, and run a supervised test of the prototype; log any failures and corrective actions.
5. Produce a final report summarizing performance, measured deviations, and next steps.

Probing Questions

- What assumptions did you make about the user's context? How could you validate them?
- Which adaptation had the biggest impact on function and why?

Quiz - Your Second Print (10 questions)

1. What is a good first question to ask a stakeholder when scoping this project? (short answer)
2. Name one safety consideration when adapting a model for daily use. (short answer)
3. What is a changelog entry? (one sentence)
4. How do you verify a fit for an assembled part? (short answer)
5. Why document corrective actions during testing? (one sentence)
6. True/False: Once you complete a successful first print, any second project will automatically succeed without iteration. (Answer: False)
7. Short answer: Describe one method to gather user feedback on your adapted model before committing to a full production print.
8. Practical scenario: Your adapted model prints but feels too fragile in daily use. What are two strategies to improve durability while maintaining the basic form?
9. Multiple choice: When you create a changelog, what should you document? (A) Only the successful changes (B) All changes, including failures and iterations (C) Only the slicer settings - Answer: B
10. Reflection: Explain how iteration (design -> print -> test -> adapt) leads to better functional outcomes than trying to get the design perfect on the first attempt. Give a specific example from your project.

Extension Problems (10)

1. Rework your prototype to improve durability and report trade-offs in weight and print time.

2. Create a user test script and run it with two participants; summarize results.
3. Convert a critical component to parametric OpenSCAD and publish the variant.
4. Add a small assembly guide with tactile cues for non-visual users.
5. Compare two filament types for the same part and recommend one with justification.
6. Execute a formal design iteration cycle: print, test, measure, analyze, revise; repeat at least 3 times and document improvements.
7. Build a comparative analysis: print your part in 2 different materials or with 2 different profiles; measure and compare all key properties.
8. Create a complete design dossier: CAD files, iteration history, test results, measurements, lessons learned, and final recommendations.
9. Develop a user testing protocol: define success metrics, recruit testers, gather systematic feedback, and iterate based on results.
10. Write a design case study: document your entire project journey from initial concept through final manufacture, including all mistakes and successes.

Your Second Print - Student Documentation Template (Extension Project)

- Author:
- Date:
- Description: Adapt or modify an existing 3D model to meet a new constraint, improve functionality, or customize for a specific use.

Ideas and Concept

Original Model

- Model name and source:
- Link or file reference:
- Why did you choose this model for adaptation?

Adaptation Goal

- What problem or constraint are you addressing?
- How will you know the adaptation is successful?
- Who will use the adapted model?

Measurements and Design Specifications

Original Model Specifications

Parameter	Value
Print time	
Filament	
Key dimension 1	
Key dimension 2	

Adapted Model Specifications

Parameter	Original	Adapted	Reason for Change

Object Notes

Modifications Made

- Describe changes to the original design:
- Did you modify the model in OpenSCAD, slicer, or both?
- Include before/after code snippets or screenshots:

Design Iteration Cycle

Print 1

- Date:
- Modifications:
- Results:
- Issues:
- Next action:

Print 2 (if applicable)

- Date:
- Modifications:
- Results:
- Issues:
- Next action:

Print 3 (if applicable)

- Date:
- Modifications:
- Results:
- Final assessment:

User Testing (if applicable)

- Who tested the adapted model?
- What feedback did they provide?
- How did this feedback influence your design?

Assembly and Use Notes

- How is the adapted model assembled or used?
- How does it differ from the original in use?
- What challenges did you encounter?

Reflections

Design Evolution

- Describe how your adaptation evolved through the iteration process
- Which iteration felt most successful?

Lessons Learned

- What did you learn about adapting existing designs?
- What surprised you during the process?

Comparison: Original vs. Adapted

- What are the key functional differences?
- Would you recommend the adapted version? Why?

Future Iterations

- If you were to continue developing this, what would you change?
- How could you make the design more parametric or reusable?

Accessibility Considerations

- How did you approach testing this project with screen readers?
- What documentation would help someone without visual access understand the changes?

Attachments

☐ Original model file (link or reference)
☐ Adapted model file (`.scad` or `.stl`)
☐ Code comparison (before/after with comments)
☐ Photos of adapted print(s)
☐ Iteration log with timestamps
☐ User feedback notes (if applicable)

☐ Slicer settings for final print

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Feedback:

Resubmission (if applicable)

What Was Changed (One-paragraph explanation of changes made and why)

Revised Score

Category	Revised Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Your Second Print - Teacher Template (Extension Project)

Briefing

Building on the first successful print, students now adapt an existing model to meet a specific constraint or improvement goal. This project emphasizes design iteration, parametric thinking, and the ability to modify existing designs for new contexts.

Key Learning: Design iteration; constraint-based modification; documentation of changes.

Real-world Connection: Adaptation and iteration are core engineering practices. Most real products are refinements of earlier versions.

Constraints

- Must be a modification or adaptation of an existing 3D model
- Modifications must be parametric (variables or commented code showing what changed)

- Student must document the modification rationale and testing process
- Iteration should be evidence through multiple print attempts or variant comparisons

Functional Requirements

- Modification is clearly documented with before/after code comparison
- Adapted print functions as intended in the modified context
- Student provides evidence of testing the adaptation
- Design shows intentional thought about materials, fit, or functionality

Deliverables

- Completed documentation template with:
  - Original model identification and link
  - Modifications made (with code comments or diff)
  - Design iteration cycle (print, test, adapt, repeat)
  - User testing results (if applicable)
  - Reflection on design decisions
  - Comparison of original vs. adapted version
- Modified `.scad` or `.stl` files showing changes
- Photos of both original and adapted prints (if possible)
- Test results or user feedback documentation

Rubric

All projects are scored on a 0-9 scale across three equally weighted categories (3 points each):

Category	Points	What We Measure
Problem & Solution	0-3	Is the adapted design functional? Does it solve the stated adaptation goal?
Design & Code Quality	0-3	Are modifications clear and well-documented? Is iteration evident? Does the part work well?
Documentation	0-3	Is the iteration cycle documented? Are design decisions explained? Is reflection thorough?

Category 1: Problem & Solution (0-3 points)

Score	Description
3	Adaptation successfully addresses design goal. Print is functional and performs as intended in modified context.
2	Adaptation mostly addresses goal. Print is functional with minor limitations.

Score	Description
1	Adaptation partially addresses goal. Print has functional limitations.
0	Adaptation does not work or is not attempted.

Category 2: Design & Code Quality (0-3 points)

Score	Description
3	Modifications clearly documented with before/after comparison. Iteration cycle evident (multiple prints/tests). Print quality excellent.
2	Modifications documented adequately. Some iteration evident. Print quality acceptable.
1	Minimal modification documentation. Little iteration. Print quality acceptable but lacks refinement.
0	Modifications not documented or design not functional.

Category 3: Documentation (0-3 points)

Score	Description
3	All sections complete. Design iteration documented with measurements and testing results. Reflection is specific. Photos included.
2	Most sections present. Iteration documented but could be more detailed. Reflection adequate.
1	Incomplete sections. Minimal iteration documentation. Reflection brief.
0	No documentation submitted.

Score Interpretation

Total Score	Interpretation	Next Step
8-9	Excellent adaptation	Student demonstrates design maturity
6-7	Good iteration and adaptation	Encourage further design work
4-5	Meets basics; improve iteration	Resubmit iteration documentation
2-3	Does not meet expectations	Resubmission or coaching
0-1	Missing components	Meet with instructor

Resubmission Policy

Students may resubmit to improve their score. Resubmissions must include:

1. A one-paragraph explanation of what was changed and why

The resubmission score replaces the original.

Assessment Notes

- Strong submissions show clear modification intent, multiple iteration cycles with documented changes, and user feedback integration
- Watch for: Minimal modifications, no iteration, or generic reflections
- Reinforce: Why iteration matters; how to document design decisions
- Extension: Portfolio development; design for manufacturability

Bonus Print - Guided Extension

Estimated time: 3-5 hours

Learning Objectives

- Apply scaling and simple modifications to an existing model
- Verify multi-part prints and manage print constraints
- Document design changes and reproduceable print settings

Materials

- Online repository access, slicer, printer, filament

Step-by-step Tasks

1. Choose a model from Thingiverse or Printables and note the original dimensions.
2. Decide on a purposeful modification (scale, add mounting holes, combine parts) and explain why.
3. Apply changes in OpenSCAD (or by scaling in the slicer) and record the new dimensions.
4. Slice and print all parts in one session when possible; log print times and filament used.
5. Create a short construction note and photograph the assembled result.

Probing Questions

- What motivated the modification and who benefits from it?
- How did scaling affect tolerances or assembly fit?

Quiz - Bonus Print (10 questions)

1. What is one risk when scaling a model up or down? (short answer)

2. Name two checks to perform before printing multi-part models. (short answer)
3. How should you record filament usage? (short answer)
4. Why document construction steps? (one sentence)
5. What is one visual sign a part needs more infill? (one sentence)
6. True/False: Scaling a model uniformly (proportionally) in all three dimensions will preserve the original fit tolerances perfectly. (Answer: False - because tolerance stack-up and printer behavior can change)
7. Short answer: Explain the difference between scaling a model in the slicer versus modifying the OpenSCAD code to scale a design. Which approach is more reproducible?
8. Practical scenario: You want to scale a model from 10 cm to 25 cm (2.5x scale). What should you check regarding print time and support material before committing to the print?
9. Multiple choice: When assembling multi-part prints, what should you test first? (A) The final assembly (B) Individual part dimensions, then pairwise assembly (C) Skip testing - Answer: B
10. Reflection: Describe how documenting your modifications (scaling factor, OpenSCAD code changes, filament type, print settings) enables other students to reproduce your design and iterate on it further.

Extension Problems (10)

1. Create a small assembly guide with tactile cues for non-visual users.
2. Produce two scaled variants and compare required print time and fit.
3. Modify a part to include snap-fit connectors and document fit tolerances.
4. Add simple labeling to parts using embossed text in OpenSCAD.
5. Publish your variant and short build notes to the class repo and review two peers' submissions.
6. Build a complete variant library: create 5+ variations of your bonus print; document parameters, reasoning, and use cases.
7. Design a variant optimization process: compare variants by cost, print time, quality, and functionality; justify your "best" choice.
8. Create a parametric master file that generates all variants automatically; test parameter ranges and edge cases.
9. Develop a variant documentation and sharing system: create a portfolio with photos, specs, and instructions for each variant.
10. Write a "remix and iterate" guide: explain how future students can modify your designs, what parameters they should change, and how to test improvements.

Bonus Print - Student Documentation Template (Extension Project)

- Author:

- Date:
- Description: Create scaled variants of a design to explore parametric thinking and manufacturing trade-offs.

Design Concept

- Base design name and description:
- Why did you choose this design?
- How will you generate variants parametrically?

Variant Specifications

Variant	Scale	Print Time (est.)	Filament (est.)	Key Dimensions
v1				
v2				
v3				

Print Results

Variant	Actual Print Time	Actual Filament	Quality	Notes
v1				
v2				
v3				

Analysis and Reflections

- Which variant is most efficient (time/material ratio)?
- How did scaling affect print quality?
- What trade-offs did you observe (speed vs. quality, material vs. time)?
- If you made a fourth variant, what would it be and why?

Attachments

☐ `.scad` file with variant generation code
☐ Photos of all variants (assembled if applicable)
☐ Print log for each variant
☐ Comparison analysis

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Bonus Print - Teacher Template (Extension Project)

Briefing

Students design scaled variants of a 3D-printable object to explore parametric design and manufacturing trade-offs. This project emphasizes variant generation, documentation, and the relationship between design parameters and print outcomes.

Key Learning: Parametric scaling; variant generation; trade-off analysis.

Real-world Connection: Manufacturing efficiency depends on understanding how parameter changes affect production time, material use, and functionality.

Constraints

- Must generate at least 3 scaled variants of a single base design
- All variants must be parametric (variables clearly documented)
- Student must document print parameters for each variant
- Variants must be compared on measurable criteria (time, material, fit, etc.)

Functional Requirements

- Variants are generated parametrically, not manually resized in slicer
- Each variant prints successfully
- Comparison metrics recorded for all variants
- Code is well-commented and organized

Deliverables

- Parametric `.scad` with variant generation code
- Completed documentation template with variant specifications
- Print log for all variants
- Comparison matrix (time, material, quality, cost)
- Photos of all variants
- Reflection on design scalability

Rubric

(Same three-category 0-9 scale as other projects)

Category 1: Problem & Solution (0-3) All variants print successfully and meet comparison criteria.

Category 2: Design & Code Quality (0-3) Code is parametric and well-commented. Variants show thoughtful design thinking.

Category 3: Documentation (0-3) Comparison matrix complete. Reflection specific and insightful.

Assessment Notes

- Strong submissions: Show parametric structure, complete comparison matrix, and insightful reflection on scaling trade-offs
- Reinforce: How parametric thinking enables rapid variant generation
- Extension: Cost-benefit analysis; material property comparisons

Lesson 3: Parametric Architecture and Modular Libraries

Estimated time: 120-150 minutes

- Define and document top-level parameters and use them to drive model variants[76]
- Create reusable modules and import a library module into a project[77]
- Produce and test low-resolution renders and export a working STL[78]

Materials

- A 3dMake project scaffold with `src/main.scad`
- Example library (e.g., BOSL) available in `lib/` or classroom assets
- Reference: openscad-cheat-sheet.md for parametric design patterns

1. Open `src/main.scad` and add three top-level parameters (with units) at the top of the file[79].

[76] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

[77] BOSL - OpenSCAD Standard Library - https://github.com/BelfrySCAD/BOSL2

[78] 3DMake GitHub - https://github.com/tdeck/3dmake

[79] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

```
// Top-level parameters - customize these
part_width = 50; // mm
part_height = 40; // mm
part_thickness = 5; // mm
```

2. Implement a simple `module bracket(width, height, thickness)` that composes primitives and boolean operations[80].

```
// Simple bracket module - combines primitives with CSG
module bracket(w, h, t) {
  union(){
    // Base plate
    cube([w, h, t], center=false);
    // Vertical support
    translate([0, h-5, 0])
      cube([t, 5, 20], center=false);
  }
}
// Use the module
bracket(part_width, part_height, part_thickness);
```

3. Include a library module (e.g., `include <bosl/constants.scad>`) and call a small helper from it; run `3dm build` [81].

   Alternative: Create your own simple library module:

```
// Create lib/helpers.scad
module simple_fillet(size, radius) {
  // Simple fillet using minkowski
  minkowski(){
    children();
    cylinder(h=0.01, r=radius, $fn=16);
  }
}
```

   Then in main.scad:

```
include <lib/helpers.scad>
simple_fillet(20, 2){
  cube([20, 20, 20]);
}
```

4. Produce a low-resolution render to confirm topology, then export an STL and inspect it in a slicer[82].

   - Use `$fn=12` or lower during development to speed up renders
   - Once satisfied, increase to `$fn=32` or `$fn=64` for final prints

5. Refactor the module into `lib/` with a short README and a usage example[83].

[80]BOSL - OpenSCAD Standard Library - https://github.com/BelfrySCAD/BOSL2
[81]BOSL - OpenSCAD Standard Library - https://github.com/BelfrySCAD/BOSL2
[82]OpenSCAD Review - Worth learning? - CadHub, accessed February 18, 2026, https://learn.cadhub.xyz/blog/openscad-review/
[83]OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

Going Deeper: Parametric Design and Modular Thinking

Now that you've created simple modules, let's explore *why* they matter for design at scale.

What Is Parametric Design? Parametric design means building your model around adjustable parameters so that a single change propagates automatically throughout your entire design. Instead of hardcoding a dimension like 50, you store it in a variable:

```
// Non-parametric (fragile):
cube([50, 50, 50]);
translate([50, 0, 0]) cube([50, 50, 50]);
translate([100, 0, 0]) cube([50, 50, 50]);
// Parametric (flexible):
box_size = 50;
cube([box_size, box_size, box_size]);
translate([box_size, 0, 0]) cube([box_size, box_size, box_size]);
translate([2*box_size, 0, 0]) cube([box_size, box_size, box_size]);
```

If you need to change the box size from 50 to 60 mm, the parametric version requires one change; the non-parametric version requires three edits.

Modules with Default Parameters A module becomes even more powerful when you define default parameters:

```
// Define a module with defaults
module storage_box(length = 40, width = 30, height = 20, wall = 2) {
    difference() {
        // Outer shell
        cube([length, width, height]);
        // Inner hollow (remove inside, keep walls and bottom)
        translate([wall, wall, wall])
            cube([length - 2*wall, width - 2*wall, height]);
    }
}
// Call with defaults
storage_box();
// Call with one parameter changed
storage_box(length = 60);
// Call with all custom parameters
storage_box(length = 80, width = 50, height = 30, wall = 3);
```

This approach creates a small, medium, and large box set:

```
// Small box (all defaults)
storage_box();
// Medium box (positioned next to small)
translate([45, 0, 0]) storage_box(length = 50, width = 40, height = 25);
// Large box (positioned next to medium)
translate([100, 0, 0]) storage_box(length = 70, width = 55, height = 35);
```

The DRY Principle: Don't Repeat Yourself When you copy-paste the same shape over and over, you create maintenance risk. If you later discover a bug in the shape, you must fix it in *every* copy. Modules eliminate this:

Without modules (DRY violation):

```
// Bracket repeated 3 times (copy-paste)
union(){ cube([10,10,10]); translate([0,20,0]) cube([2,30,10]); }
translate([50, 0, 0]) union(){ cube([10,10,10]); translate([0,20,0]) cube →
    ↪ ([2,30,10]); }
translate([100, 0, 0]) union(){ cube([10,10,10]); translate([0,20,0]) cube →
    ↪ ([2,30,10]); }
```

With modules (DRY compliant):

```
module bracket() {
    union(){
        cube([10,10,10]);
        translate([0,20,0]) cube([2,30,10]);
    }
}
bracket();
translate([50, 0, 0]) bracket();
translate([100, 0, 0]) bracket();
```

Scope and Reusability Parameters are scoped to their module, meaning you can't access a module's parameters from outside it. This is intentional-it prevents naming conflicts and makes modules truly reusable:

```
module box(size = 10) {
    cube([size, size, size]);
}
// You can use 'size' in multiple contexts without conflict
size = 100; // Global size
box(); // Calls box with its default size=10
translate([size, 0, 0]) box(size = 50); // Calls box with size=50
```

Importing Libraries: `include` vs `use` Once you've written modules in `lib/helpers.scad`, you can share them with other projects:

```
// Option 1: include (gives you all functions and modules)
include <lib/helpers.scad>
// Option 2: use (brings in functions but hides module definitions)
use <lib/helpers.scad>
// Now you can call any module from helpers.scad
simple_fillet(20, 2) {
    cube([20, 20, 20]);
}
```

For most classroom use, `include` is simpler to understand. For larger projects, `use` provides cleaner namespace isolation.

Testing Your Modules Before committing a module to your library, test it with:

1. Different parameter values: Call the module with minimum, maximum, and mid-range values
2. Default values: Verify that calling it with no parameters produces a reasonable result
3. Rendered output: Use `$fn = 32` to render the module and visually inspect for geometry issues

4. STL export: Export to STL and open in your slicer to check for non-manifold warnings

Checkpoints

- After step 2 you have a parametric module you can call with different arguments[84].
- After this section, you should understand how to create modules with default parameters and why that matters for reusability.

Advanced Technique: Loops and Repetition

Beyond creating individual modules, OpenSCAD allows you to create multiple copies using for loops. This is powerful for creating arrays, spirals, or radially symmetric designs.

Example: Creating Arrays with For Loops

The `for()` function iterates over a range and creates multiple copies of geometry:

```
// Simple 3x3 array of cubes
$fn = 32;
for (x = [0:20:40])
  for (y = [0:20:40])
    translate([x, y, 0])
      cube([10, 10, 10]);
```

What it does:

- `x = [0:20:40]` creates x positions: 0, 20, 40
- Nested `y` loop creates 9 positions total (3 x 3)
- Each cube is translated to its position

Practical Example: Parametric Fin Can

This real-world example from the *Simplifying 3D Printing* textbook demonstrates using for loops to create radially symmetric fins around a cylinder:

```
// Parametric Fin Can - demonstrates for loops and radial arrays
$fn = 100;
// Parameters
height = 70;
diameter = 43.5;
thickness = 1.2;
guide_diameter = 3;
fins = 3; // Number of fins to create
// Define the fin profile as a 2D polygon
fin_shape = [[0, 0], [height, 0], [100, 35], [100, 45], [35, 35]];
// Module to create a single fin
module create_fin(angle = 0) {
    rotate([0, 0, angle])
```

[84] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

```
    translate([diameter/2, 0, height])
    rotate([90, 90, 0])
    linear_extrude(height = thickness, center = true)
      polygon(fin_shape);
}
// Main assembly
module generate_fin_can() {
    difference() {
        union() {
            // Central cylinder
            cylinder(h = height, d = diameter + thickness);
            // Create fins using a for loop
            for (i = [0 : fins - 1]) {
                create_fin(i * (360 / fins));
            }
            // Guide post
            rotate([0, 0, 180 / fins])
            translate([diameter/2 + guide_diameter/2 + thickness/2, 0, 0])
            difference() {
                cylinder(h = height, d = guide_diameter + thickness);
                cylinder(h = height, d = guide_diameter);
            }
        }
        // Hollow center
        cylinder(h = 1000, d = diameter, center = true);
    }
}
// Render the complete assembly
generate_fin_can();
```

Key points:

- The `for (i = [0 : fins - 1])` loop creates a fin at each angle: 0deg, 120deg, 240deg
- The angle is calculated as `i * (360 / fins)` to distribute fins evenly
- Each iteration calls `create_fin()` with a different rotation angle
- This creates 3 identical fins perfectly positioned without copying code

Common For Loop Patterns

Pattern	Use Case	Example
`for (x = [0:10:50])`	Linear array	Grid of objects
`for (i = [0:n-1])`	Index-based loop	Radial array with calculated angles
Nested for loops	2D grids	Multiple rows and columns
`for (i = [start:step:end])`	Custom spacing	Items at irregular intervals

Try This: Modify the Fin Can

```
// Experiment with changing these parameters:
fins = 4; // Try 4, 5, or 6 fins
height = 100; // Make it taller
diameter = 60; // Make it wider
thickness = 2; // Make fins thicker
// The for loop automatically distributes the fins evenly around the cylinder
```

Checkpoint: After this section, you can:

- Use `for` loops to create arrays of objects
- Calculate rotation angles for radial symmetry
- Combine loops with modules to create parametric families of designs

Advanced Programming Concepts

Beyond basic modules, professional OpenSCAD code uses conditional logic, type checking, and advanced data structures. These techniques enable robust, flexible, and maintainable designs.

1. Conditional Logic with If-Else Statements

Use `if/else` to create designs that adapt based on parameters. This enables single-code designs that behave differently based on user input:

```
// Parametric phone stand with optional features
phone_width = 75;
include_cable_holder = true; // Toggle feature with parameter
module phone_stand() {
  // Always create the base
  cube([phone_width + 20, 100, 5]);
  // Conditionally add cable holder
  if (include_cable_holder) {
    translate([phone_width + 10, 20, 5])
      cube([10, 15, 20]); // Cable management slot
  }
}
phone_stand();
```

Practical Example: Parametric Bracket with Optional Features

```
// Advanced bracket with conditional features
bracket_size = 50;
include_mounting_holes = true;
include_reinforcement = true;
hole_diameter = 3;
module bracket() {
  // Main body (always created)
  difference() {
    cube([bracket_size, bracket_size, 10]);
    // Hollow interior
    translate([5, 5, 5])
      cube([bracket_size - 10, bracket_size - 10, 10]);
  }
  // Optional: Add mounting holes
  if (include_mounting_holes) {
    for (x = [10, bracket_size - 10])
      for (y = [10, bracket_size - 10])
        translate([x, y, -5])
          cylinder(h=20, d=hole_diameter, $fn=16);
  }
  // Optional: Add diagonal reinforcement
  if (include_reinforcement) {
    rotate([45, 0, 0])
      cube([bracket_size - 10, 5, 5], center=true);
  }
}
bracket();
```

Key Patterns:

```
// Simple conditional
if (parameter) {
  // Create this if parameter is true
  cube([10, 10, 10]);
}
// If-else conditional
if (parameter > 50) {
  // Create large version
  cube([100, 100, 100]);
} else {
  // Create small version
  cube([50, 50, 50]);
}
// Multiple conditions
if (enable_feature_a && enable_feature_b) {
  // Both features enabled
} else if (enable_feature_a) {
  // Only A enabled
} else if (enable_feature_b) {
  // Only B enabled
} else {
  // Neither enabled
}
```

2. Type Testing and Validation

Before using parameters, test their type to avoid errors. This makes modules robust and user-friendly:

```
// Type testing functions: is_number(), is_string(), is_list(), is_bool(), is_undef()
// Example: Create a function that accepts either a single number or list
module create_array(size) {
  if (is_number(size)) {
    // Single number: create 3x3 array of that size
    for (x = [0:20:40])
      for (y = [0:20:40])
        translate([x, y, 0])
          cube([size, size, size]);
  } else if (is_list(size)) {
    // List: [width, depth, height]
    for (x = [0:20:40])
      for (y = [0:20:40])
        translate([x, y, 0])
          cube(size); // Use list directly
  } else {
    // Error: invalid type
    echo("ERROR: size must be number or list");
  }
}
// Both work:
create_array(10); // Creates 10x10x10 cubes
create_array([15, 15, 20]); // Creates 15x15x20 cubes
```

Practical Validation Example:

```
// Robust parametric keycap with validation
module keycap(size, height, letter) {
  // Validate inputs before using
  valid_size = is_number(size) && size > 5 && size < 50;
  valid_height = is_number(height) && height > 3 && height < 20;
  valid_letter = is_string(letter) && len(letter) == 1;
  if (valid_size && valid_height && valid_letter) {
    // All parameters valid: create keycap
    difference() {
      cube([size, size, height]);
      translate([size/2, size/2, size/2 - 1])
        cube([size - 4, size - 4, height], center=true);
    }
    // Add embossed letter
    translate([size/2, size/2, height - 0.5])
      linear_extrude(0.8)
        text(letter, size=size*0.4, halign="center", valign="center");
  } else {
    // Parameter validation failed: report errors
    if (!valid_size) echo("ERROR: size must be 5-50mm");
    if (!valid_height) echo("ERROR: height must be 3-20mm");
    if (!valid_letter) echo("ERROR: letter must be single character");
  }
}
// These work:
keycap(14, 10, "A");
// These will fail gracefully with error messages:
// keycap(-5, 10, "A"); // Invalid size
// keycap(14, 10, "ABC"); // Invalid letter
```

3. List Comprehensions and Advanced Data Structures

List comprehensions create new lists by transforming or filtering existing lists. This enables sophisticated parametric designs:

```
// Basic list comprehension syntax: [for (x = list) ... x ... ]
// Example 1: Create list of squares
squares = [for (x = [1:5]) x*x];
// Result: [1, 4, 9, 16, 25]
// Example 2: Filter list (keep only even numbers)
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
evens = [for (x = numbers) if (x % 2 == 0) x];
// Result: [2, 4, 6, 8, 10]
// Example 3: Create array of positions for mounting holes
hole_positions = [for (i = [0:3]) [i*20, i*20, 0]];
// Creates: [[0,0,0], [20,20,0], [40,40,0], [60,60,0]]
```

Practical Example: Parametric Grid of Features

```
// Create parametric grid of mounting holes using list comprehension
bracket_width = 100;
bracket_depth = 80;
hole_spacing = 20; // mm between holes
hole_diameter = 3; // mm
// Generate hole positions automatically
hole_x_positions = [for (i = [0:(bracket_width/hole_spacing)]) i *
    hole_spacing];
hole_y_positions = [for (i = [0:(bracket_depth/hole_spacing)]) i *
    hole_spacing];
module bracket_with_holes() {
  difference() {
    // Base plate
    cube([bracket_width, bracket_depth, 5]);
    // Subtract holes using comprehension
    for (x = hole_x_positions)
      for (y = hole_y_positions)
        translate([x, y, -5])
          cylinder(h=15, d=hole_diameter, $fn=16);
  }
}
bracket_with_holes();
```

4. Special Variables: $preview, $children, $fn

OpenSCAD includes special variables that give modules information about their context:

```
// $preview: Detects if in preview (F5) or render (F6) mode
module adaptive_quality() {
  quality = $preview ? 16 : 64; // Low res in preview, high in render
  for (i = [0:360/quality:360])
    rotate([0, 0, i])
      translate([50, 0, 0])
        sphere(r=5, $fn=quality);
}
// $children: Number of child objects passed to module
module repeat_with_offset(count) {
  for (i = [0:count-1])
    translate([i*20, 0, 0])
      children(); // Render each child at new position
}
// Usage:
repeat_with_offset(3)
  cube([10, 10, 10]);
// Creates 3 cubes in a row, spaced 20mm apart
// $fn, $fa, $fs: Global resolution parameters
// $fn = 32; // Number of segments (0 disables, uses $fa/$fs)
// $fa = 12; // Angle per segment (degrees)
// $fs = 2; // Minimum segment length (mm)
```

Practical Example: Adaptive Quality Module

```
// Module that renders efficiently in preview but beautifully in export
module rounded_box(w, h, d, radius) {
  // Automatically adjust quality based on render mode
  fn = $preview ? 8 : 32; // Fast preview, quality export
  minkowski() {
    cube([w - 2*radius, h - 2*radius, d - 2*radius], center=true);
    cylinder(r=radius, h=0.01, $fn=fn);
  }
}
// Use normally; quality automatically adapts
rounded_box(50, 50, 50, 3);
```

5. User-Defined Functions vs Modules

OpenSCAD supports both modules (which create geometry) and functions (which compute values). Understanding the difference is critical:

```
// MODULE: Creates 3D geometry (what we've been using)
module box(w, h, d) {
  cube([w, h, d]);
}
// FUNCTION: Computes and returns a value (no geometry)
function box_volume(w, h, d) = w * h * d;
// Usage:
box(50, 50, 50); // Creates 50x50x50 cube
volume = box_volume(50, 50, 50); // Calculates: 125000
echo(volume); // Prints: 125000
// ============================================
// Practical Example: Recursive function for power calculation
function power(base, exp) =
  exp == 0 ? 1 : // Base case: anything^0 = 1
  base * power(base, exp - 1); // Recursive case
// Calculate 2^5
result = power(2, 5);
```

```
echo(result); // Prints: 32
// ============================================
// Another Example: Distance between two points
function distance(p1, p2) =
  sqrt((p2[0] - p1[0])^2 + (p2[1] - p1[1])^2 + (p2[2] - p1[2])^2);
point_a = [0, 0, 0];
point_b = [3, 4, 0];
dist = distance(point_a, point_b);
echo(dist); // Prints: 5 (Pythagorean triple)
```

Key Differences:

Feature	Module	Function
Creates geometry	Yes	No
Returns value	No	Yes
Can use `children()`	Yes	No
Can combine with echo/if	Yes	Yes
Use for	3D objects	Calculations

Practical Combined Example:

```
// Use function to calculate, module to create
// Function: Calculate optimal wall thickness based on size
function optimal_wall_thickness(size) =
  size < 20 ? 1.0 :
  size < 50 ? 1.5 :
  size < 100 ? 2.0 :
  2.5;
// Module: Create parametric container
module container(width, height) {
  wall = optimal_wall_thickness(width);
  difference() {
    cube([width, width, height]);
    translate([wall, wall, wall])
      cube([width - 2*wall, width - 2*wall, height]);
  }
}
// Usage
container(30, 40); // 1mm walls automatically
container(75, 100); // 1.5mm walls automatically
container(120, 150); // 2mm walls automatically
```

Quiz - Lesson 3dMake.3 (10 questions)

1. What is a parametric module and why is it useful[85]?
2. How do you include an external library in OpenSCAD[86]?
3. What is one advantage of moving code into `lib/` [87]?
4. How can you test a module quickly without a full high-resolution render[88]?

[85] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

[86] BOSL - OpenSCAD Standard Library - https://github.com/BelfrySCAD/BOSL2

[87] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

[88] OpenSCAD Review - Worth learning? - CadHub, accessed February 18, 2026, https://learn.cadhub.xyz/blog/openscad-review/

5. Name one safety or documentation step to include when producing a reusable module[89].
6. True or False: Parametric modules can only accept one parameter.
7. Describe what ”Don't Repeat Yourself” (DRY) means in the context of 3D model code.
8. Explain how you would create a module with three parameters and use it in your main file.
9. What testing strategies should you use to validate a new module before using it in production prints?
10. How would you document a reusable module so that future users (including yourself) understand its parameters and limitations?

Extension Problems (10)

1. Create a module for a mounting bracket with parameters for hole size and spacing; publish a usage example[90].
2. Use a library module to add a fillet and compare the final STL before and after[91].
3. Produce three variants by changing a parameter and record estimated print times[92].
4. Move a working module into `lib/` and commit with a clear commit message[93].
5. Write a short README for your module describing parameter ranges and expected units[94].
6. Build a complete module library with 5+ reusable parts; create comprehensive documentation and usage examples for each.
7. Design a module versioning and compatibility system; document how to manage parameter changes over time.
8. Create a module testing suite: write validation tests for parameter ranges, edge cases, and output consistency.
9. Develop an accessibility guide for module parameters: explain how to use and understand module options without visual feedback.
10. Write a module library contribution guide: standards for code style, documentation, testing, and review for collaborative development.

---

[89] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

[90] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

[91] BOSL - OpenSCAD Standard Library - https://github.com/BelfrySCAD/BOSL2

[92] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

[93] 3DMake GitHub - https://github.com/tdeck/3dmake

[94] OpenSCAD Modules - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules

Supplemental Resources

For deeper exploration of parametric design and module creation, explore these resources:

- Programming with OpenSCAD EPUB Textbook - Comprehensive guide to modules, parametric design, and library creation
- CodeSolutions: Modules - Working examples of module definition, parameter passing, and reusable components
- BOSL2 Library - Professional OpenSCAD library with hundreds of reusable modules for common operations

Starter Templates

Parametric Template

```
// Parametric Design Template
// Demonstrates how to create designs that adapt to different inputs
// This pattern allows you to quickly generate variations
// ============================================
// CUSTOMIZABLE PARAMETERS
// ============================================
// Box dimensions (change these to regenerate different sizes)
boxwidth = 40;
boxdepth = 40;
boxheight = 30;
// Feature sizing (scales with box size)
cornerradius = 3;
wallthickness = 2;
// Boolean flags for feature toggling
includelid = true;
includefeet = true;
hollowinterior = true;
// ============================================
// QUALITY SETTINGS
// ============================================
$fn = 30; // Fragment quality
// ============================================
// PARAMETRIC MODULES
// ============================================
// Simple box with rounded corners
module roundedbox(width, depth, height, radius) {
    hull() {
        // Create 8 corner spheres to define the shape
        translate([radius, radius, radius])
            sphere(r = radius);
        translate([width - radius, radius, radius])
            sphere(r = radius);
        translate([radius, depth - radius, radius])
            sphere(r = radius);
        translate([width - radius, depth - radius, radius])
            sphere(r = radius);
        translate([radius, radius, height - radius])
            sphere(r = radius);
        translate([width - radius, radius, height - radius])
            sphere(r = radius);
        translate([radius, depth - radius, height - radius])
            sphere(r = radius);
        translate([width - radius, depth - radius, height - radius])
            sphere(r = radius);
    }
}
// Use the module
roundedbox(boxwidth, boxdepth, boxheight, cornerradius);
```

Module Template

```
// Module Design Pattern Template
// Demonstrates best practices for organizing code with reusable modules
// Modules promote code reuse, clarity, and maintainability
// ============================================
// CONFIGURATION SECTION
// ============================================
$fn = 32;
// Component dimensions
connectordiameter = 8;
connectorlength = 15;
connectorwall = 1.5;
basesize = 50;
baseheight = 5;
// ============================================
// UTILITY MODULES (General Purpose)
// ============================================
// Create a hollow cylinder (tube/connector)
module hollowcylinder(od, id, height) {
    difference() {
        cylinder(h = height, d = od, center = true);
        cylinder(h = height + 1, d = id, center = true);
    }
}
// Create a flat base/mounting plate
module flatbase(size, thickness) {
    cube([size, size, thickness], center = true);
}
// Create corner supports for strengthening
module cornersupport(size, height) {
    linearextrude(height = height, center = true)
        polygon(points = [[0, 0], [size, 0], [0, size]]);
}
// ============================================
// ASSEMBLY
// ============================================
// Combine modules for a complete design
union(){
    flatbase(basesize, baseheight);
    translate([0, 0, baseheight])
        hollowcylinder(connectordiameter, connectordiameter - 2*connectorwall →
            ↪ , connectorlength);
}
```

Lesson 4: AI-Enhanced Verification and Multimodal Feedback

Estimated time: 45-60 minutes

Learning Objectives

- Run `3dm info` to collect deterministic renders and AI diagnostics (if configured)[95][96]
- Compare AI suggestions with renderer and slicer outputs and prioritize deterministic fixes[97]

[95] 3DMake GitHub - https://github.com/tdeck/3dmake
[96] Deterministic Validation in Design - https://www.nist.gov/publications
[97] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

- Record and sanitize AI prompts and outputs for reproducibility and data governance[98]

Materials

- 3dMake configured with an LLM API key (optional)
- Example project with renderable geometry

Step-by-step Tasks

1. Verify API configuration (if used) or run `3dm info --dry-run` to confirm render pipeline works locally[99].
2. Run `3dm info` and save the produced images and textual report to `build/`[100].
3. Inspect deterministic outputs (render warnings, slicer preview) and compare them to AI recommendations; prioritize deterministic issues[101][102].
4. Iterate prompt engineering (in `3dmake.toml` or via `--prompt`) with precise technical primitives and re-run `3dm info` to examine changes[103].
5. Document all prompts, AI outputs, and deterministic validation steps in `AI-notes.md` within the project[104].

Checkpoints

- After step 2 you have stored render images and the AI report in `build/`[105].

Quiz - Lesson 3dMake.4 (10 questions)

1. What command generates AI diagnostics and model renders[106]?
2. Why must AI outputs be validated against renderer/slicer results[107][108]?
3. Name one privacy or governance concern when sending models/images to an API[109].

[98] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[99] 3DMake GitHub - https://github.com/tdeck/3dmake

[100] 3DMake GitHub - https://github.com/tdeck/3dmake

[101] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[102] Deterministic Validation in Design - https://www.nist.gov/publications

[103] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[104] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[105] 3DMake GitHub - https://github.com/tdeck/3dmake

[106] 3DMake GitHub - https://github.com/tdeck/3dmake

[107] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[108] Deterministic Validation in Design - https://www.nist.gov/publications

[109] AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

4. What is an example of a technical primitive to include in a prompt[110]?
5. Where should you record prompts and AI outputs in the project[111]?
6. True or False: AI-generated suggestions should always be implemented without verification.
7. Describe what ”deterministic validation” means in the context of AI-assisted 3D design.
8. Explain how you would use `3dm info` to validate AI suggestions against your actual model.
9. What steps would you take if the AI suggests a design change that conflicts with your requirements?
10. How would you document AI-assisted decisions in your project for reproducibility and transparency?

Extension Problems (10)

1. Create an `AI-notes.md` documenting three prompts and the AI's responses; indicate which suggestions you acted on[112].
2. Simulate a false-positive AI warning: describe how you validated and rejected it using deterministic checks[113].
3. Generate a short prompt that requests the top three structural risks and record the results[114].
4. Create a short checklist for sanitizing uploads before sending to an API[115].
5. Re-run `3dm info` after a code fix and compare the differences in the AI report[116].
6. Design a verification workflow: use AI suggestions + manual validation + physical testing to validate design decisions.
7. Create a transparency log for AI-assisted features: document all AI interactions, suggestions adopted, and decisions made.
8. Build an AI-assisted design case study: from initial prompt through final print; document the entire decision process.
9. Develop guidelines for trustworthy AI use in 3D printing: when to trust AI, when to verify, and how to validate recommendations.
10. Write an accessibility guide for using AI features in 3DMake: how to interpret AI suggestions non-visually and understand confidence levels.

[110]AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[111]AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[112]AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[113]Deterministic Validation in Design - https://www.nist.gov/publications

[114]AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[115]AI Output Validation - Best Practices for Prompt Engineering - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

[116]3DMake GitHub - https://github.com/tdeck/3dmake

Dice Design Workshop - Guided Extension

Estimated time: 3-6 hours

Learning Objectives

- Design parametric dice in OpenSCAD with controlled mass distribution
- Test durability and randomness for small, thrown objects
- Document design decisions and measure physical outcomes

Materials

- Computer with OpenSCAD and slicer
- Printer and filament, small testing surface

Step-by-step Tasks

1. Create three distinct die designs in OpenSCAD using parameters (size, face style, internal cavities).
2. For each die, explain which parameters you changed and why (one short paragraph each).
3. Print one sample of each die and perform a basic durability test (five throws onto a soft surface).
4. Record outcomes of each throw and compute a simple frequency table for face results.
5. Measure mass and critical dimensions; document any deformation or failure modes.

Probing Questions

- How does internal infill or cavity affect mass distribution and roll randomness?
- Which parameter had the largest effect on durability and why?

Quick Quiz (10 questions)

1. What parameter controls wall thickness in your `.scad` file? (short answer)
2. Name one method to increase a part's impact resistance. (short answer)
3. How would you test randomness in a die? (short answer)
4. Why might a hollow die behave differently than a solid one? (one sentence)
5. What is a simple sign of print-layer delamination? (one sentence)
6. True/False: A perfectly fair die must have uniform weight distribution throughout, including the internal structure. (Answer: True)
7. Short answer: Describe two design approaches to a die that could be parametrically scaled - one using `cube()` with `difference()` for texture, one using `linear_extrude()` for embossed numbers.

8. Practical scenario: Your first die printed, but when tested with 100 rolls, it landed on face #1 significantly more often than other faces. What are two possible causes and how would you diagnose the problem?
9. Multiple choice: If you increase infill from 15% to 50%, what happens? (A) Dramatically longer print time and higher weight (B) Slightly longer print time, much higher weight (C) Same weight, different durability - Answer: B
10. Reflection: Explain why a fair die is an excellent test of design precision and iterative improvement. How does this project teach you about manufacturability, quality assurance, and design validation?

Extension Problems (10)

1. Create a parametric script that produces dice with adjustable center-of-mass offsets and show results of roll tests.
2. Design a bead-style die that assembles from two printed halves and document assembly steps.
3. Compare two infill patterns for the same die and report on mass and durability differences.
4. Add tactile markers to faces for non-visual testing and document how they affect roll behavior.
5. Publish your `.scad` and test logs to the class repo and provide feedback on two classmates' designs.
6. Conduct a formal fairness test: roll your die 100+ times, track results, and perform statistical analysis to verify randomness.
7. Design multiple die variants (sizes, infills, materials); compare manufacturing feasibility, cost, and performance for each.
8. Create a parametric die library: build a system that generates dice of different sizes, materials, and properties on command.
9. Develop a comprehensive quality assurance protocol: define tolerances, measurement methods, and acceptance criteria for fair dice.
10. Write a die design guide for future students: explain the physics of fair dice, how to test for bias, and how to iterate on designs.

Dice, Dice, Dice - Student Documentation Template (Extension Project)

- Author:
- Date:
- Description: Design parametric 3D-printed dice and test for fairness through statistical analysis.

Design Concept

- Dice specifications (size, material, infill):
- How will you ensure uniform weight distribution?

- What design details will your dice include (pips, patterns)?

Design Specifications

Parameter	Value
Die size	
Infill %	
Material	
Pip depth	
Pit/pip count per face	

Fairness Testing

Test Setup

- How many rolls will you conduct? (minimum 100)
- Rolling surface and method:
- Date/time of testing:

Roll Results

Face	Frequency (out of _ rolls)	Percentage	Expected %
1			16.67%
2			16.67%
3			16.67%
4			16.67%
5			16.67%
6			16.67%

Statistical Analysis

- Calculate chi-squared value or deviation from expected:
- Is the die fair (within statistical tolerance)?
- What face appears most/least frequently?

Reflections

- Were you surprised by the results?
- What design factors influenced fairness?
- How would you improve the dice design?
- What did you learn about manufacturing precision?

Attachments

- ☐ `.scad` file with parametric dice module
- ☐ Photo of test dice
- ☐ Roll data (raw results or tally sheet)
- ☐ Statistical analysis calculations
- ☐ Design iteration notes (if multiple versions tested)

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Dice, Dice, Dice - Teacher Template (Extension Project)

Briefing

Students design and test fair 3D-printed dice to explore geometry, symmetry, manufacturing precision, and statistical validation. This project combines physics, materials science, and rigorous testing practices.

Key Learning: Precision manufacturing; fairness validation; quality assurance.

Real-world Connection: Manufacturing quality assurance requires measurement, testing, and statistical analysis. Casino dice are one of the most rigorously tested manufactured objects.

Constraints

- Dice must be generated parametrically in OpenSCAD
- Student must conduct fairness testing (100+ rolls minimum)
- Design must address weight distribution and uniform geometry
- Statistical analysis of test results required

Functional Requirements

- Dice geometry is parametric (size, infill, details)
- All faces print with usable pips/markers
- Statistical testing conducted with documented results
- Analysis shows understanding of fairness criteria

Deliverables

- `.scad` file with parametric dice module
- Completed documentation template with fairness test results
- Statistical analysis (roll frequency data, calculations)
- Photos of test dice and assembly
- Reflection on design precision and manufacturing tolerances

Rubric

Category 1: Problem & Solution (0-3) Dice print successfully and fairness testing conducted rigorously.

Category 2: Design & Code Quality (0-3) Code is parametric and thoughtful. Dice design shows understanding of symmetry and balance.

Category 3: Documentation (0-3) Testing data complete and organized. Statistical analysis accurate and insightful.

Assessment Notes

- Strong submissions: Show rigorous fairness testing (100+ rolls), clear statistical analysis, and reflection on manufacturing precision
- Reinforce: Why uniformity matters; how to test for bias
- Extension: Material comparisons; cost-benefit analysis of infill vs. fairness

Lesson 5: Safety Protocols and the Physical Fabrication Interface

Estimated time: 60-90 minutes

Learning Objectives

- Describe the Hierarchy of Controls and apply it to a classroom makerspace[117][118]
- Perform pre-print environmental and equipment checks[119]
- Validate the digital-to-physical pipeline and document post-print inspection results[120][121]

[117] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/

[118] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad

[119] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/

[120] STL Validation and Geometry Checking - https://docs.prusa3d.com/en/guide/39012-validation-tools/

[121] Post-Print Inspection Procedures - https://www.prusa3d.com/support/

Materials

- Classroom 3D printer, enclosure, and filtration (or documented lab SOP)
- Example parametric OpenSCAD project and slicer profile

Step-by-step Tasks

1. Conduct a safety briefing: review the Hierarchy of Controls and locate safety equipment[122].
2. Verify environmental controls and printer readiness: confirm filtration, bed adhesion, and spool metadata[123][124].
3. Run `3dm build` and inspect the STL in a slicer; check layer-preview and non-manifold warnings[125][126].
4. If using remote submission (OctoPrint), confirm camera monitoring and logging before any unattended prints[127].
5. Obtain instructor sign-off and monitor the first layers in-person or via camera.
6. After printing, wait for the part to cool (< $30^\circ\text{C}$), measure critical dimensions, and record observations[128][129].

Checkpoints

- Completed environmental checklist and saved inspection notes in the project README[130][131].

Introduction to Material Selection

Before your part goes to print, it's important to understand how material choice affects the printing process, the final product's properties, and the safety considerations for your workspace.

Why Material Choice Matters Most projects in this class use PLA - and for good reason. It's the easiest, cheapest, and lowest-emission option. But as you advance, you'll encounter projects that require different materials:

- A flexible phone case needs TPU (thermoplastic polyurethane)

[122] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[123] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[124] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad
[125] STL Validation and Geometry Checking - https://docs.prusa3d.com/en/guide/39012-validation-tools/
[126] Post-Print Inspection Procedures - https://www.prusa3d.com/support/
[127] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[128] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[129] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad
[130] STL Validation and Geometry Checking - https://docs.prusa3d.com/en/guide/39012-validation-tools/
[131] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/

- A functional bracket that will be outdoors might use PETG (polyethylene terephthalate)
- A decorative miniature works perfectly in PLA

Three Common Materials: Quick Reference Each material has different print temperatures, strength properties, and difficulty levels. Here's a quick comparison:

Material	Print Temp	Difficulty	Best Uses	Cautions
PLA	190-220C	Easy	Prototypes, classroom projects, decorative models	Brittle; low heat resistance; low emission
PETG	230-250C	Moderate	Functional parts, mechanical components, water resistance	Stringing issues; requires bed heat; more emission than PLA
TPU	220-240C	Hard	Flexible parts, phone cases, gaskets, wearables	Requires slow speeds; produces more fumes; needs ventilation

How Material Choice Affects Your Slicer Settings When you switch materials, you must adjust:

- Nozzle Temperature: Too cold and plastic won't extrude; too hot and it burns
- Bed Temperature: Helps adhesion; critical for PETG, optional for PLA
- Print Speed: TPU requires very slow speeds (20-30 mm/s) to prevent extrusion problems
- Cooling: PLA benefits from active cooling; TPU does not

Most slicers (like PrusaSlicer) have material profiles that automatically set these values. Always verify the temperatures match your filament's specifications (check the spool label).

Environmental and Safety Considerations Different materials produce different levels of emission:

- PLA: Lowest concern; generally safe in ventilated rooms
- PETG: Low emission; still requires good ventilation
- TPU: Moderate to higher emission; requires active ventilation or filtration

Always follow your classroom's safety protocol and consult Appendix B: Material Properties & Selection for a comprehensive reference on all available materials, their specifications, and best practices.

Looking Ahead: Material and Design Decisions Later in this curriculum, you'll encounter projects that specifically require:

- Flexible materials (Lesson 8: Assembly and Durability)
- High-strength materials (Lesson 10: Quality Assurance and Measurement)
- Multiple-material assemblies (Lesson 11: Stakeholder-Centric Design)

By understanding material properties now, you'll be prepared to make informed design decisions in those lessons.

Quiz - Lesson 3dMake.5 (10 questions)

1. What are the four levels of the Hierarchy of Controls[132]?
2. Name two engineering controls useful for reducing emissions[133][134].
3. Why must you monitor the first layers of a new print profile[135]?
4. Where should you record spool metadata and print observations[136][137]?
5. What is the safe cooldown temperature suggested before part removal[138]?
6. True or False: Ventilation is only necessary for commercial printing facilities.
7. Describe the relationship between print temperature, material type, and emissions.
8. Explain what "spool metadata" includes and why tracking it is important for reproducibility.
9. How would you measure and record critical dimensions on a printed part?
10. What health and safety precautions should be documented in an SOP for supervised prints?

Extension Problems (10)

1. Draft a one-page SOP for start-to-finish supervised prints in your lab[139][140].
2. Create a checklist script that verifies spool metadata and build settings before `3dm build` runs[141].
3. Run a test print and log measured deviations; propose a parameter change

[132] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[133] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[134] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad
[135] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[136] STL Validation and Geometry Checking - https://docs.prusa3d.com/en/guide/39012-validation-tools/
[137] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[138] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad
[139] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/
[140] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad
[141] STL Validation and Geometry Checking - https://docs.prusa3d.com/en/guide/39012-validation-tools/

to correct the error[142].

4. Design an accessible post-print inspection checklist that non-visual users can follow[143][144].
5. Research filtration options and recommend one for your classroom, including maintenance intervals[145][146].
6. Develop a comprehensive classroom safety and operations manual: SOPs, checklists, emergency procedures, and accessibility considerations.
7. Create a print quality assurance system: define metrics, measurement methods, and acceptance criteria for finished prints.
8. Build a calibration and maintenance log: document all calibration activities, maintenance dates, and performance metrics over time.
9. Design a student training certification program: define knowledge and skill requirements, create assessments, and track competency.
10. Write an operations handbook that covers setup, troubleshooting, safety, maintenance, and accessibility for your 3D printing facility.

Safety Checklist for 3D Printing

Complete this checklist before each printing session to maintain a safe workspace.

Pre-Print Setup

☐ Work Area Clear

- Clear desk/table of unnecessary items
- Remove tripping hazards around printer
- Ensure adequate ventilation around the printer

☐ Printer Inspection

- Check nozzle is clean (no old filament residue)
- Verify build plate is level
- Confirm heated bed temperature sensor attached
- Check all cables are secure and not frayed

☐ Filament Preparation

- Filament spool rotates freely (no binding)
- Filament path clear of obstructions
- Extruder drive gear clean (not clogged with plastic)

[142] Post-Print Inspection Procedures - https://www.prusa3d.com/support/

[143] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/

[144] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad

[145] ANSI/AIHA Z590.3 - Hierarchy of Controls - https://www.aiha.org/

[146] 10+ OpenSCAD Online Courses for 2026 | Explore Free Courses & Certifications, accessed February 18, 2026, https://www.classcentral.com/subject/openscad

- New filament loaded correctly

☐ Environmental Conditions

- Room temperature adequate (18-25C ideal)
- No direct drafts from windows/AC on printer
- Humidity levels reasonable (not extremely dry)
- Adequate lighting to see printer clearly

During Print

☐ First Layer Monitoring

- Watch first 2-3 layers closely
- Check bed adhesion (not too loose or tight)
- Confirm nozzle temperature stable
- Monitor for any unusual sounds

☐ Regular Checks

- Check print every 15-30 minutes initially
- Verify filament is feeding smoothly
- Listen for grinding or skipping noises
- Watch for layer shifting or warping

☐ Temperature Stability

- Heated bed maintains consistent temperature
- Nozzle temperature doesn't fluctuate
- No thermal runaway warnings

Safety Alerts

☐ STOP Print Immediately If

- Nozzle jams or extrudes unevenly
- Filament completely stops feeding
- Burning smell detected
- Visible layer shifting
- Extruder grinding or skipping sounds
- Any unusual smells (especially chemical/burning)
- Power indicators show failure

Post-Print

☐ Cool Down Period

- Allow heated bed to cool naturally (10-15 min)
- Keep hands clear until nozzle cools
- Verify printer is idle before touching

☐ Print Removal

- Use proper tools (spatula/scraper)
- Remove print when bed fully cooled
- Inspect print for defects or sharp edges
- Sand sharp edges if necessary

☐ Equipment Cleaning

- Wipe nozzle with brass wire when cooled
- Remove any plastic debris from build plate
- Check extruder for filament residue
- Clean any visible dust from electronics

☐ Workspace Cleanup

- Return tools to proper storage
- Dispose of support material safely
- Tidy workspace and verify clear pathways
- Store filament properly in dry location

Hazard Awareness

Hot Surfaces:

- Nozzle reaches 200-250C
- Bed reaches 60-100C
- Allow adequate cooling time before touching

Filament Hazards:

- Tangled filament can jam printer
- Always keep filament spool free-spinning
- Check for knots or kinks before printing

Electrical Safety:

- Never operate with wet hands
- Keep liquids away from printer
- Unplug before major maintenance
- Verify power supply is grounded

Material Hazards:

- Some filaments emit fumes when heated
- Ensure adequate ventilation
- Use HEPA filter if printing indoors
- Dispose of failed prints properly

Emergency Contacts

- Printer Fire: Unplug immediately, use dry chemical extinguisher (NOT water)
- Burns: Rinse with cool water for 15+ minutes
- Inhalation Issues: Move to fresh air, seek medical attention if symptoms persist

Last Reviewed: _
Reviewed By: _
Notes:

Printer Maintenance Log

Track routine maintenance, repairs, and operational issues to keep your printer running smoothly.

Printer Information

- Model: _
- Serial Number: _
- Purchase Date: _
- Last Service Date: _

Maintenance Schedule

Daily (Before Each Use)

☐ Visual inspection for damage
☐ Clean nozzle if needed
☐ Check build plate level
☐ Verify filament spool rotates freely

Weekly

☐ Clean extruder drive gear
☐ Inspect and clean Z-axis rails
☐ Check all cable connections
☐ Test emergency stop function

Monthly

☐ Full build plate leveling procedure
☐ Clean interior of printer chamber
☐ Inspect heating elements
☐ Test temperature calibration

Quarterly (Every 3 Months)

- ☐ Replace worn nozzle if needed
- ☐ Full mechanical inspection
- ☐ Software/firmware update check
- ☐ Complete system test print

Maintenance Log

Date	Maintenance Type	Description	Time Spent	Issues Found	Resolution	Performed By

Issue Tracking

Date	Symptom	Diagnosis	Action Taken	Status	Notes

Parts Replacement

Date	Part	Reason	Supplier	Cost	Notes

Filament Compatibility Notes

Test results for different filament types used with this printer:

Filament Type	Brand	Nozzle Temp	Bed Temp	Print Speed	Success Rate	Notes
PLA						
PETG						
ABS						
TPU						
Other:						

Troubleshooting Reference

Common Issues and Solutions Nozzle Clogs:

- Date First Noticed: _
- Attempts to Clear: _
- Successful Solution: _

Bed Adhesion Problems:

- Suspected Cause: _
- Solution Applied: _
- Result: _

Layer Shifting:

- Frequency: _
- Likely Causes: _
- Actions Taken: _

Extrusion Issues:

- Symptoms: _
- Diagnosed Problem: _
- Fix Applied: _

Printer Performance Metrics

Track overall printer health over time:

Month	Print Success Rate	Avg Print Time	Common Issues	Overall Assessment

Warranty & Support Information

- Warranty Expires: _
- Manufacturer Support: _
- Local Technician: _
- Emergency Contact: _

Last Log Entry: _
Logged By: _

Lesson 6: Practical 3dm Commands and Text Embossing

Estimated time: 90-120 minutes

Learning Objectives

- Master key 3dMake commands: `3dm describe`, `3dm preview`, `3dm orient`, `3dm slice` [147]
- Apply text embossing and parametric customization to create a personalized keycap[148]
- Generate tactile 2D previews for accessible model inspection[149]

Materials

- 3dMake project scaffold with `src/main.scad`
- Access to a slicer (PrusaSlicer, Cura, or Bambu Studio)
- Optional: printer for test print
- Reference: 3dmake-setup-guide.md for command reference

Related Project: See cube_keycap.scad for a worked example combining text embossing and the 3dm commands.

Understanding the 3dm Command Suite

The 3dMake CLI provides a systematic workflow for model management. Each command serves a specific phase of the design-to-print pipeline.

Command	Purpose	Example
`3dm describe <file>`	Generates text and AI analysis of geometry	`3dm describe src/main.scad`
`3dm preview <file>`	Creates a 2D tactile printout (ASCII/braille/SVG)	`3dm preview src/main.scad`

[147] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[148] OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
[149] 3DMake GitHub Repository - https://github.com/tdeck/3dmake

Command	Purpose	Example
`3dm orient <file>`	Analyzes and suggests optimal print orientation	`3dm orient src/main.scad`
`3dm slice <file>`	Generates G-code (machine instructions)	`3dm slice src/main.scad`

Step-by-step Tasks

1. Create a parametric keycap model with top-level parameters for customization[150]:

```
// Cube Keycap - Parametric Design
// A customizable keycap with embossed text

// Parameters (easy customization)
key_size = 18; // mm - width/depth of keycap
key_height = 12; // mm - height
wall = 1.2; // mm - wall thickness
letter = "A"; // Character to emboss
letter_size = 10; // mm - text height
letter_raise = 0.8; // mm - emboss depth

// Hollow shell
module shell(){
  difference(){
    cube([key_size, key_size, key_height], center=false);
    translate([wall, wall, wall])
      cube([key_size-2*wall, key_size-2*wall, key_height], center=
          false);
  }
}

// Text embossing on top face
module emboss(){
  translate([key_size/2, key_size/2, key_height-0.01])
    linear_extrude(height=letter_raise)
      text(letter, size=letter_size, halign="center", valign="center"
          );
}

// Assemble keycap
union(){
  shell();
  emboss();
}
```

2. Run `3dm describe` to generate a text-based analysis of your model:

```
3dm describe src/main.scad
```

   This will output a description suitable for non-visual inspection. If AI is configured, you'll also receive suggestions about geometry and potential print issues.

3. Generate a tactile 2D preview with `3dm preview`:

[150] OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text

```
3dm preview src/main.scad
```

This creates a simplified 2D outline that can be printed for physical inspection or converted to braille for accessibility.

4. Analyze print orientation with `3dm orient`:

```
3dm orient src/main.scad
```

This command suggests the optimal orientation for printing-minimizing support material and maximizing surface quality.

5. Prepare for printing by slicing your model:

```
3dm slice src/main.scad
```

This generates G-code (`build/main.gcode`) ready for your printer.

String Functions for Dynamic Naming and Labeling

OpenSCAD includes functions for manipulating strings, enabling designs that automatically label themselves, create dynamic descriptions, or adapt text based on parameters.

String Concatenation: str()

Combine strings and numbers into single strings:

```
// Basic concatenation
name = str("Keycap_", "Large");
echo(name); // Prints: Keycap_Large
// Combine strings and numbers
size = 25;
label = str("Size: ", size, "mm");
echo(label); // Prints: Size: 25mm
// Complex concatenation
width = 50;
height = 30;
material = "PLA";
description = str(material, " box ", width, "x", height);
echo(description); // Prints: PLA box 50x30
```

Practical Example: Self-Labeling Container

```
// Container that labels itself with dimensions
module labeled_container(w, h, d, material_type) {
  // Create dimension label
  label = str(w, "x", h, "x", d, " ", material_type);
  // Create container body
  difference() {
    cube([w, h, d]);
    translate([5, 5, 5])
      cube([w-10, h-10, d-5]);
  }
  // Add embossed label on side
  translate([w/2, -2, d/2])
    linear_extrude(0.8)
    text(label, size=3, halign="center", valign="center", font="Arial:style
        =Bold");
}
labeled_container(60, 40, 30, "PLA");
// Output: Creates box labeled "60x40x30 PLA"
```

String Length and Searching: len(), search()

Find and measure strings:

```
// len(): String length
echo(len("OpenSCAD")); // Prints: 8
echo(len("Hi")); // Prints: 2
// search(): Find substring
text = "manufacturing";
result = search("fact", text);
echo(result); // Returns index where "fact" is found
```

Practical Example: Smart Keycap Text

```
// Keycap that validates text before embossing
module smart_keycap(key_size, text_to_engrave) {
  text_length = len(text_to_engrave);
  // Choose appropriate text size based on length
  text_size = text_length > 3 ? 8 :
             text_length > 1 ? 12 :
             14; // Single character gets larger
  echo(str("Engraving '", text_to_engrave, "' at ", text_size, "mm"));
  difference() {
    cube([key_size, key_size, 10]);
    translate([2, 2, 2])
      cube([key_size-4, key_size-4, 10]);
  }
  // Emboss with auto-scaled text
  translate([key_size/2, key_size/2, 9.5])
    linear_extrude(0.8)
      text(text_to_engrave, size=text_size, halign="center", valign="center")
        ;
}
smart_keycap(18, "A"); // Single letter: 14mm
smart_keycap(18, "Ctrl"); // Multi-char: 8mm
```

Substring Extraction: substr()

Extract parts of strings:

```
// substr(): Extract substring
original = "OpenSCAD";
first_three = substr(original, 0, 3);
echo(first_three); // Prints: Ope
last_three = substr(original, 5, 3);
echo(last_three); // Prints: CAD
```

Practical Example: Serial Number Generation

```
// Generate sequential serial numbers
module part_with_serial(part_name, serial_number) {
  // Create formatted serial: PART_001, PART_002, etc.
  serial_string = str(
    substr(part_name, 0, 4), // First 4 letters of part_name
    "_",
    serial_number < 10 ? str("00", serial_number) :
```

```
    serial_number < 100 ? str("0", serial_number) :
    str(serial_number)
  );
  echo(str("Creating: ", serial_string));
  // Create part body
  cube([20, 20, 15]);
  // Add serial number embossing
  translate([10, 10, 14.5])
    linear_extrude(0.5)
      text(serial_string, size=2, halign="center", valign="center", font=" →
           ↪ Arial");
}
part_with_serial("bearing", 1); // Output: BEAR_001
part_with_serial("bearing", 27); // Output: BEAR_027
part_with_serial("bearing", 150); // Output: BEAR_150
```

Variable Scoping with let()

Use `let()` to create temporary local variables in expressions:

```
// let() creates scoped variables in expressions
radius = 10;
// Without let: must use full expression
area = 3.14159 * radius * radius;
// With let: clearer and reusable
area = let(r = radius, pi = 3.14159) pi * r * r;
// Practical: Complex calculations
box_volume = let(
  w = 50,
  h = 30,
  d = 20
) w * h * d;
echo(box_volume); // Prints: 30000
```

Practical Example: Parametric Gear with Formatted Output

```
// Complete gear design with dynamic labeling
module parametric_gear(
  pitch_diameter,
  teeth_count,
  bore_diameter,
  gear_name
) {
  // Use let() for intermediate calculations
  calculations = let(
    module = pitch_diameter / teeth_count,
    addendum = 1.0,
    dedendum = 1.25,
    outer_d = pitch_diameter + 2 * addendum,
    root_d = pitch_diameter - 2 * dedendum
  ) [module, outer_d, root_d];
  // Create formatted specification
  spec = str(
    gear_name, ": ",
    teeth_count, " teeth, ",
    pitch_diameter, "mm PD"
  );
  echo(spec);
  // Create gear body
  difference() {
    cylinder(r=calculations[1]/2, h=8, $fn=teeth_count*2);
    cylinder(r=bore_diameter/2, h=10);
  }
}
parametric_gear(40, 20, 8, "InputGear");
// Output: "InputGear: 20 teeth, 40mm PD"
```

Combining String Functions for Batch Documentation

Here's a complete example generating documentation for multiple variants:

```
// Generate a family of keycaps with automatic documentation
module keycap_family() {
  variants = [
    ["S", 12, 8], // [letter, size, height]
    ["M", 16, 10],
    ["L", 20, 12],
    ["XL", 24, 14]
  ];
  for (v = variants) {
    letter = v[0];
    size = v[1];
    height = v[2];
    // Build specification string
    spec = str(
      "Keycap_", letter, " (",
      size, "x", size, "x", height, "mm)"
    );
    echo(spec);
    // Position variant for display
    x_offset = (letter == "S") ? 0 :
               (letter == "M") ? 30 :
               (letter == "L") ? 60 :
               90;
    translate([x_offset, 0, 0]) {
      difference() {
        cube([size, size, height]);
        translate([1.5, 1.5, 1.5])
          cube([size-3, size-3, height-1.5]);
      }
      // Emboss the letter
      translate([size/2, size/2, height - 0.5])
        linear_extrude(0.8)
          text(letter, size=size*0.5, halign="center", valign="center");
    }
  }
}
keycap_family();
// Output:
// Keycap_S (12x12x8mm)
// Keycap_M (16x16x10mm)
// Keycap_L (20x20x12mm)
// Keycap_XL (24x24x14mm)
```

Checkpoint

- After task 1, you have a working keycap model with parameters
- After task 3, you have a 2D tactile preview in `build/`
- After task 5, you have G-code ready to send to a printer

Customization Challenge

Modify the keycap by changing these parameters and observing the results:

Parameter	Original	Try This	Effect
`key_size`	18	25	Larger keycap
`key_height`	12	8	Shorter, lower-profile key

Parameter	Original	Try This	Effect
`letter`	”A”	”YOUR_INITIAL”	Different character
`wall`	1.2	1.5	Thicker walls (stronger)
`letter_raise`	0.8	1.5	Deeper emboss

Quiz - Lesson 3dMake.6 (10 questions)

1. What does `3dm describe` provide that `3dm preview` does not[151]?
2. Why is `3dm preview` useful for accessibility[152]?
3. What does `3dm orient` help you determine about your model[153]?
4. How does parametric design (using variables) make customization easier[154]?
5. Name one use case for embossed text in a 3D model[155].
6. True or False: You must run `3dm build` before running `3dm describe`.
7. Explain the relationship between wall thickness and print strength.
8. Describe what happens when you change the `$fn` parameter in your text() function.
9. How would you modify the keycap code to add a second embossed character[156]?
10. What information would you include in a `3dm describe` output for a manufacturer[157]?

Extension Problems (10)

1. Create three keycap variants (small, medium, large) by changing `key_size` and export each as a separate STL[158].
2. Add a decorative border using `linear_extrude` and `text()`; verify the result with `3dm preview` [159].
3. Run `3dm describe`, `3dm preview`, and `3dm orient` on your keycap; document all outputs[160].

[151] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[152] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[153] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[154] OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
[155] OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
[156] OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
[157] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[158] OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
[159] 3DMake GitHub Repository - https://github.com/tdeck/3dmake
[160] 3DMake GitHub Repository - https://github.com/tdeck/3dmake

4. Modify the emboss module to support multiple characters in different positions[161].
5. Create a parametric keycap library with variants for different keyboard layouts (cherry, OEM, DSA profiles)[162].
6. Design an accessibility assessment: use `3dm describe` and `3dm preview` to validate your model for non-visual users.
7. Build a custom keycap design system: parameterize profile, height, wall thickness, and emboss content; generate a product family.
8. Create a comparison study: test three different emboss depths and document print quality differences.
9. Develop a keycap troubleshooting guide: common failures (emboss erosion, wall cracking, text illegibility) and solutions.
10. Write a comprehensive keycap design documentation: parameters, materials, print settings, post-processing, and accessibility features.

References

Slicing Settings Quick Reference - PrusaSlicer

Recommended Settings by Use Case

Use Case	Layer Height	Infill	Supports	Notes
Quick test / prototype	0.30 mm	10%	As needed	”Draft” - fastest, roughest
Standard project	0.20 mm	15-20%	As needed	Best all-around starting point
Functional part	0.20 mm	30-40%	As needed	Use for parts under stress
Fine detail / display	0.15 mm	15%	As needed	Smoother surface; slower
Solid reference part	0.20 mm	40-50%	Rarely	Rarely needed; long print

Filament Temperature Settings

Filament	Nozzle Temp	Bed Temp	Notes
PLA	200-215C	50-60C	Easiest to print; default choice

[161]OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
[162]OpenSCAD Manual - Text & Linear Extrude - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text

Filament	Nozzle Temp	Bed Temp	Notes
PETG	230-250C	70-85C	Use glue stick on PEI bed
TPU	220-240C	30-60C	Print at 20-30 mm/s max; direct drive only
ABS	230-250C	90-110C	Requires enclosure; more fumes

Always check the temperature range on your filament spool - it may vary by brand.

Support Settings Guide

Overhang Angle	Supports Needed?	Recommended Setting
< 45	No	None
45-60	Maybe	Preview first; add if sagging
> 60	Yes	Support on build plate only
Bridge < 20 mm	No	Bridges usually fine
Bridge > 20 mm	Maybe	Preview; consider reorienting

Common Print Problems & Quick Fixes

Problem	Likely Cause	Fix
Print lifts off bed	Poor adhesion / warping	Add brim; use glue stick; level bed
Stringing between parts	Temperature too high / retraction	Lower temp 5C; check retraction settings
Layer lines very visible	Layer height too thick	Use 0.15mm or 0.20mm
Print takes too long	Layer height too thin / infill too high	Use 0.30mm draft; reduce infill
Holes too small	FDM tolerance - always undersized	Add 0.2-0.3mm to hole diameter
Part broke at layer boundary	Weak axis perpendicular to layers	Reorient so load is parallel to layers
First layer not sticking	Bed not level	Run bed leveling routine
Spaghetti / print failure	No supports on overhang	Add supports or reorient

G-code Export Checklist

Before exporting, confirm:

- ☐ Correct printer profile selected
- ☐ Correct filament profile selected
- ☐ Layer height appropriate for use case
- ☐ Infill percentage set
- ☐ Supports enabled if needed
- ☐ Layer preview reviewed (no floating parts, supports where needed)
- ☐ Print time and filament weight noted for your records

Sources

Prusa Research. (2023). *PrusaSlicer knowledge base*. https://help.prusa3d.com/category/prusaslicer_204
Hubs. (2023). *What is FDM 3D printing?* https://www.hubs.com/knowledge-base/what-is-fdm-3d-printing/
ThePrusaSlicer.net. (2025). *How to use PrusaSlicer*. https://theprusaslicer.net/how-to-use-prusaslicer/

Parametric Keychain - Extension Project

Estimated time: 2-4 hours

Learning Objectives

By completing this project, you will:

- Create parametric OpenSCAD modules that accept user inputs
- Implement 2D text manipulation and 3D extrusion techniques
- Generate and test multiple design variants systematically
- Document design parameters for reproducibility and user customization

Objective

- Create a parametric keychain design that adapts to custom text, dimensions, and materials.

Tasks

1. Create `keychain.scad` with top-level parameters: `width`, `height`, `thickness`, and `text`.
2. Implement embossed or debossed text using `linear_extrude()` of a 2D text shape (or simulate with simple geometry if system lacks text support).
3. Produce three size variants and export STLs; record print settings.
4. Test attachment point for common key rings and report fit.

Deliverable

- Source `keychain.scad` file with parametric variables documented
- Three STL variants (small, medium, large)
- Print settings log and fit-test report for key ring attachment

Starter files

- starter.scad - minimal parametric scaffold to begin.

Starter Code

Copy and modify this cube keycap example as your starting point:

```
// Cube Keycap - Beginner Example
// A simple 20mm cube keycap with an embossed letter
// Parameters
keysize = 18; // mm
keyheight = 12; // mm
wall = 1.2; // mm
letter = "R"; // change to your preferred letter
lettersize = 10; // mm
letterraise = 0.8;// mm
module shell(){
  difference(){
    cube([keysize, keysize, keyheight], center=false);
    translate([wall, wall, wall])
      cube([keysize-2*wall, keysize-2*wall, keyheight], center=false);
  }
}
module emboss(){
  // Emboss letter on top face
  translate([keysize/2, keysize/2, keyheight-0.01])
    linearextrude(height=letterraise)
      text(letter, size=lettersize, halign="center", valign="center");
}
union(){
  shell();
  emboss();
}
```

Advanced Text Techniques

Beyond simple embossed letters, you can create sophisticated text effects using the patterns from the *Simplifying 3D Printing* textbook:

Example 1: Circular Text Array Arrange text in a circle around a central point:

```
// Circular Text Arrangement
// Text rotates around a center point
module rotate_text(display_text,
    text_size = 10,
    distance = 20,
    rotation_value = 360,
    tilt = 0)
{
    rotate([0, 0, tilt])
    for(i = [0:len(display_text) - 1])
    {
        rotate([0, 0, -i * rotation_value / len(display_text)])
        translate([0, distance, 0])
        text(display_text[i],
            font = "Impact:style=Regular",
            size = text_size,
            halign = "center");
    }
}
// Use the module to create circular text
linear_extrude(height = 2)
rotate_text("MAKER", text_size = 12, distance = 30, rotation_value = 75,  →
    ↪ tilt = 30);
```

What it does:

- Rotates each letter individually around a center point
- Creates circular or spiral text effects
- Useful for badges, nameplates, and decorative objects

Example 2: Multi-Line Text Composition Combine text at different sizes and positions for a professional nameplate:

```
// Professional Nameplate with Multiple Text Layers
module create_nameplate(name, role, company)
{
    union()
    {
        // Base backing plate
        cube([120, 60, 3], center = true);
        // Main name - large, centered
        translate([0, 15, 2])
        linear_extrude(height = 2)
        text(name, size = 24, font = "Impact:style=Regular",
            halign = "center", valign = "center");
        // Role - medium, slightly smaller
        translate([0, 0, 2])
        linear_extrude(height = 2)
        text(role, size = 14, font = "Arial:style=Regular",
            halign = "center", valign = "center");
        // Company name - small, bottom
        translate([0, -15, 2])
        linear_extrude(height = 2)
        text(company, size = 10, font = "Arial:style=Regular",
            halign = "center", valign = "center");
    }
}
// Create a custom nameplate
create_nameplate("Alex Chen", "3D Design Engineer", "MakerCorp");
```

Key features:

- Multiple text elements at different scales
- Layered composition for professional appearance
- Each text element can be customized independently

Example 3: Parametric Font Selection Use different fonts for different effects:

```
// Different fonts create different aesthetics
$fn = 100;
// Impact font (bold, modern)
```

```
translate([0, 40, 0])
linear_extrude(height = 2)
text("BOLD", size = 20, font = "Impact:style=Regular", halign = "center");
// Arial font (clean, professional)
translate([0, 10, 0])
linear_extrude(height = 2)
text("Clean", size = 20, font = "Arial:style=Regular", halign = "center");
// Script-like (decorative)
translate([0, -20, 0])
linear_extrude(height = 2)
text("Script", size = 20, font = "Courier:style=Regular", halign = "center") →
    ↪ ;
```

Font options (availability depends on your system):

- `Impact:style=Regular` - Bold, condensed, modern
- `Arial:style=Regular` - Clean, neutral, professional
- `Courier:style=Regular` - Monospace, technical
- System fonts vary; check OpenSCAD documentation for your platform

Practical Project: Custom Keychain Nameplate Combine everything into a professional keychains with multiple variants:

```
// Customizable Keychain Nameplate
// Parameters - change these to create variants
name = "ALEX";
keysize = 35;
keyheight = 8;
wall = 1.2;
textsize = 16;
module keychain_nameplate(name, width, height, textsize) {
   union() {
      // Shell
      difference() {
         cube([width, width, height], center = false);
         translate([wall, wall, wall])
            cube([width - 2*wall, width - 2*wall, height], center = false);
      }
      // Embossed name
      translate([width/2, width/2, height - 0.01])
      linear_extrude(height = 0.8)
      text(name, size = textsize, font = "Impact:style=Regular",
          halign = "center", valign = "center");
      // Attachment loop
      translate([width/2, -3, height/2])
      rotate([90, 0, 0])
      cylinder(d = 8, h = 6, center = true, $fn = 32);
   }
}
// Create the nameplate
keychain_nameplate(name, keysize, keyheight, textsize);
```

Assessment Questions (Optional)

1. How did you use OpenSCAD parameters to enable users to customize the keychain without editing code?
2. What were the key differences in print time and material usage between your three variants?
3. Describe how you tested the key ring attachment and what adjustments you would make for the final design.

Parametric Keychain - Student Documentation Template (Extension Project)

- Author:
- Date:
- Description: Design a fully parametric keychain that supports personalization and customization.

Design Concept

- Keychain theme or purpose:
- Design elements (shape, attachment, personalization method):
- Parametric strategy (what will be variables?):

Parametric Variables

Variable	Default Value	Purpose

Variant Configurations

Variant	Parameter 1	Parameter 2	Parameter 3	Notes
v1				
v2				
v3				

Print and Assembly Results

- Describe how each variant prints:
- Assembly/attachment method:
- Functionality (does it work as intended?):

Reflections

- Which parameterization strategy was most effective?
- How could someone else customize this design?
- What would you add in a future iteration?

Customization Guide

- Clear instructions for modifying parameters
- Examples of common customizations
- Best practices for variant generation

Attachments

☐ `.scad` file with full parametric structure
☐ Photos of 2+ printed variants
☐ Variant specification table
☐ Usage/customization guide

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Parametric Keychain - Teacher Template (Extension Project)

Briefing

Students design a personalized keychain using fully parametric OpenSCAD code. This project emphasizes parameter-driven design, customization, and user-centered iteration.

Key Learning: Parametric modularity; customization; design for manufacturing variation.

Real-world Connection: Mass customization is a growing manufacturing trend. Parametric design enables efficient production of personalized products.

Constraints

- Keychain must be fully parametric (text, size, material all variables)
- Design must include at least one test for different parameter values
- Assembly instructions must support varied configurations
- Code must be documented for future customization

Functional Requirements

- All design elements are parametric variables
- Keychain includes personalization (names, initials, dates)

- Multiple variant configurations tested and documented
- Design is reproducible and shareable

Deliverables

- `.scad` with parametric keychain module
- Variant specification table (3+ configurations)
- Photos of at least 2 printed variants
- Customization guide for future users
- Code documentation and usage examples

Rubric

Category 1: Problem & Solution (0-3) Keychains print successfully and are functional/customizable.

Category 2: Design & Code Quality (0-3) Code is fully parametric and well-organized. Variants work well.

Category 3: Documentation (0-3) Variant table complete. Customization guide clear and detailed.

Assessment Notes

- Strong submissions: Show comprehensive parametric thinking, multiple tested variants, and clear guidance for customization
- Reinforce: Documentation for design reuse
- Extension: User feedback on customization preferences; commercial potential analysis

Lesson 7: Parametric Transforms and the Phone Stand Project

Estimated time: 105-135 minutes

Learning Objectives

- Use parametric transforms (`rotate()`, `translate()`, `scale()`) to position and orient sub-assemblies[163]
- Apply the Minkowski operation as a method for creating filleted edges[164]

[163]OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

[164]OpenSCAD Manual - Minkowski - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski

- Create multi-part assemblies where each component serves a distinct structural function[165]
- Test and validate parametric variations before printing[166]

Materials

- 3dMake project scaffold with `src/main.scad`
- Access to a printer or slicing software
- Measuring tools (calipers) for post-print validation

Related Project: Study phone_stand.scad for an advanced example combining multiple transforms and Minkowski operations in a real-world assembly.

Understanding Parametric Transforms

Transforms are the foundation of positioning objects in 3D space. Unlike drag-and-drop interfaces, OpenSCAD requires you to explicitly specify every position and rotation. This precision is what enables parametric design-once defined mathematically, a model can be infinitely reconfigured.

Core Transform Operations

Operation	Syntax	Example
Translate (move)	`translate([x, y, z]) { ... }`	`translate([10, 0, 0]) cube([5, 5, 5]);`
Rotate	`rotate([x_deg, y_deg, z_deg]) { ... }`	`rotate([45, 0, 0]) cube([5, 5, 5]);`
Scale	`scale([x, y, z]) { ... }`	`scale([2, 1, 1]) cube([5, 5, 5]);`
Minkowski (fillets/rounding)	`minkowski() { shape; fillet; }`	`minkowski(){ cube([10,10,10]); cylinder(r=2, h=0.01); }`

Translating Objects: Moving in 3D Space

`translate([x, y, z])` moves an object in three-dimensional space. The key insight is that you specify the movement *before* creating or referencing the object:

[165] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md
[166] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

```
// Move a cube 30 mm to the right
translate([30, 0, 0]) cube([20, 20, 20]);
// Move it up by 10 mm
translate([0, 0, 10]) cube([20, 20, 20]);
// Move it diagonally (right and forward)
translate([30, 20, 0]) cube([20, 20, 20]);
```

A critical concept: When you use `translate()`, OpenSCAD moves the *coordinate system*, not just the object. The object is then created in the new coordinate system. This means:

```
// These are functionally equivalent:
translate([10, 0, 0]) cube([20, 20, 20]);
cube([20, 20, 20]);
translate([10, 0, 0]) cube([20, 20, 20]); // Second cube shifted right
```

For multi-part assemblies, you typically nest `translate()` calls within modules:

```
module phone_stand() {
  // Base stays at origin
  base();
  // Back support is translated up and back
  translate([0, -30, base_thickness])
    rotate([65, 0, 0])
      back_support();
}
```

Rotating Objects: Orientation Around Axes

`rotate([x_deg, y_deg, z_deg])` rotates an object around the X, Y, and Z axes (in degrees):

```
// Rotate 45 around X axis (tilts forward/back)
rotate([45, 0, 0]) cube([10, 10, 10]);
// Rotate 90 around Y axis (rotates left/right)
rotate([0, 90, 0]) cube([10, 10, 10]);
// Rotate 45 around Z axis (spins in place)
rotate([0, 0, 45]) cube([10, 10, 10]);
// Rotate around all three axes
rotate([45, 30, 15]) cube([10, 10, 10]);
```

Order matters: When you specify multiple rotations, they are applied in sequence (X, then Y, then Z). This can produce unexpected results:

```
// These produce different final orientations:
rotate([45, 90, 0]) cube([10, 10, 10]);
rotate([90, 45, 0]) cube([10, 10, 10]);
```

Combining Transforms: The Order of Operations

You can nest transforms to build complex positions. Remember: OpenSCAD applies transforms from the inside out:

```
// Example: Position a cylinder so it sticks up from the back of a base
// Step 1: Create a cylinder at the origin
cylinder(r=5, h=20, $fn=32);
```

```
// Step 2: Translate it to the right spot
translate([0, 0, base_thickness])
  cylinder(r=5, h=20, $fn=32);
// Step 3: In a module, combine positioning
module mounting_peg() {
  translate([base_width/2, base_depth - 10, base_thickness])
    cylinder(r=5, h=20, $fn=32);
}
// Inside a phone_stand() assembly:
mounting_peg();
```

Practical Tip: Coordinate System Visualization

When working with complex assemblies, it helps to visualize the coordinate system:

```
// Small axes indicator (add to your design temporarily for debugging):
module axes() {
  color("red") cube([20, 1, 1]); // X axis (red)
  color("green") cube([1, 20, 1]); // Y axis (green)
  color("blue") cube([1, 1, 20]); // Z axis (blue)
}
// Place it at key points to verify positioning:
axes();
translate([100, 0, 0]) axes(); // Check alignment at offset
```

Step-by-step Tasks

Task 1: Build a Simple Phone Stand Base

Create a parametric base plate that can be adjusted for different phone weights and sizes:

```
// Phone Stand - Base Component
// Adjustable platform for holding phones and tablets
// === TOP-LEVEL PARAMETERS (customize these) ===
base_width = 70; // mm - front-to-back width
base_depth = 90; // mm - side-to-side depth
base_thickness = 4; // mm - thickness of base
angle = 65; // degrees - tilt angle
lip_height = 12; // mm - height of friction lip
fillet_r = 6; // mm - edge rounding radius
// === MODULES ===
// Simple rectangular plate
module plate(w, d, t) {
  cube([w, d, t], center=false);
}
// Fillet edges using Minkowski sum (approximation)
module filleted_plate(w, d, t, r) {
  minkowski() {
    plate(w, d, t);
    cylinder(h=0.01, r=r, $fn=40);
  }
}
// Base of the stand
module base() {
  translate([0, 0, 0])
    filleted_plate(base_width, base_depth, base_thickness, fillet_r);
}
// Back support angled for viewing
module back() {
  // Rotate the plate to create the angle
  rotate([angle, 0, 0])
    filleted_plate(base_width, base_depth, base_thickness, fillet_r);
}
// Friction lip to prevent phone from sliding
module lip() {
  translate([0, base_depth - 8, base_thickness])
    cube([base_width, 8, lip_height], center=false);
}
// === ASSEMBLE ===
union() {
  base();
  back();
  lip();
}
```

Task 2: Test Parameter Variations

Save your file and test each variant by modifying the parameters and running `3dm build`:

```
# Build the base version
3dm build

# Then try modifying angle in main.scad and rebuild
# angle = 45; // Shallow angle for viewing documents
# 3dm build

# Or try a steep angle for portrait viewing
# angle = 75; // Steep for reading
# 3dm build
```

Document the impact:

Parameter	Value	Use Case	Print Time
`angle`	45	Shallow viewing (documents, web browsing)	~1.5 hrs
`angle`	65	Comfortable video watching	~1.5 hrs
`angle`	75	Steep vertical viewing	~1.5 hrs

Task 3: Run `3dm orient` to Optimize Orientation

```
3dm orient src/main.scad
```

This command analyzes your model and suggests:

- Optimal rotation for minimal support material
- Estimated support volume that will need to be removed
- Print time savings from better orientation

Task 4: Generate Variants for Different Devices

Modify your main.scad to create three configurations (tablet, phone, document holder):

```
// At the bottom of main.scad, uncomment one configuration:
// Configuration 1: Phone (narrow, shallow angle)
// base_width = 60;
// angle = 55;
// Configuration 2: Tablet (wide, moderate angle)
base_width = 120;
angle = 40;
// Configuration 3: Document (wide, steep angle)
// base_width = 200;
// angle = 20;
```

Task 5: Validate and Document

After printing (or slicing), record:

- Actual dimensions (measure with calipers)
- Angle accuracy (verify tilt angle with protractor or phone measurement app)
- Friction resistance (does phone stay in place?)
- Print quality (note any support marks, layer quality)

Advanced: Adding Snap-Fit Connectors

To join the base and back without fasteners, you can add interlocking features:

```
// Optional: Add snap-fit connectors
// Slot in base plate (where back plate slides in)
module base_slot() {
  slot_width = base_thickness + 0.5; // Slight clearance
  slot_depth = 20;
  translate([base_width/2 - slot_width/2, 0, base_thickness])
    cube([slot_width, slot_depth, lip_height]);
}
// Tab on back plate (fits into base slot)
module back_tab() {
  tab_width = base_thickness;
  tab_height = lip_height;
  translate([base_width/2 - tab_width/2, 0, 0])
    cube([tab_width, 20, tab_height]);
}
```

Checkpoint

- After task 1, you have a working 3-part stand (base, back, lip)
- After task 2, you've tested at least 2 parameter variations
- After task 4, you have 3 different configurations ready to print

Mathematical Functions for Advanced Parametric Design

Beyond translate and rotate, OpenSCAD includes mathematical functions for calculating positions, angles, and dimensions. These enable designs that respond dynamically to parameters.

Trigonometric Functions: sin(), cos(), atan2()

Trigonometric functions calculate angles and positions in circular patterns. Essential for gears, spirals, and polar arrangements:

```
// Place objects in circular pattern using sin/cos
module circular_array(radius, count) {
  for (i = [0:count-1]) {
    angle = (360 / count) * i;
    x = radius * cos(angle);
    y = radius * sin(angle);
    translate([x, y, 0])
      children();
  }
}
// Create circular array of 8 cylinders
circular_array(30, 8)
  cylinder(h=10, r=3);
// Spiral staircase using sin/cos
module spiral_staircase(radius, height, steps) {
  step_height = height / steps;
  step_angle = 360 / steps;
  for (i = [0:steps-1]) {
    angle = i * step_angle;
    z = i * step_height;
    x = radius * cos(angle);
    y = radius * sin(angle);
    translate([x, y, z])
      cube([20, 5, step_height]);
  }
}
spiral_staircase(40, 100, 20);
```

Practical Example: Gear Tooth Positioning

```
// Simple gear using sin/cos for tooth placement
module simple_gear(radius, teeth_count, tooth_depth) {
  tooth_angle = 360 / teeth_count;
  difference() {
    // Main gear body
    cylinder(r=radius, h=5, $fn=128);
    // Center hole
    cylinder(r=radius*0.3, h=10);
  }
  // Add teeth using circular pattern
  for (i = [0:teeth_count-1]) {
    angle = i * tooth_angle;
    x = (radius + tooth_depth) * cos(angle);
    y = (radius + tooth_depth) * sin(angle);
    z = 5/2;
    translate([x, y, z])
      cube([5, 5, 5], center=true);
  }
}
simple_gear(30, 12, 5); // 30mm radius, 12 teeth, 5mm deep
```

Rounding Functions: round(), floor(), ceil()

Control precision and create integer-based parametric designs:

```
// round(): Nearest integer
echo(round(3.7)); // Prints: 4
echo(round(3.2)); // Prints: 3
// floor(): Round down
echo(floor(3.9)); // Prints: 3
```

```
// ceil(): Round up
echo(ceil(3.1)); // Prints: 4
// Practical use: Ensure grid alignment
grid_spacing = 7.3;
aligned_spacing = round(grid_spacing); // 7
```

Practical Example: Grid-Aligned Mounting Pattern

```
// Create mounting pattern aligned to 5mm grid
module grid_mounting_pattern(width, height, pattern_spacing) {
  // Round spacing to nearest 5mm for printability
  aligned_spacing = round(pattern_spacing / 5) * 5;
  // Calculate how many holes fit
  holes_x = floor(width / aligned_spacing);
  holes_y = floor(height / aligned_spacing);
  echo(str("Creating grid: ", holes_x, " x ", holes_y, " at ", →
      ↪ aligned_spacing, "mm spacing"));
  for (x = [0:holes_x-1])
    for (y = [0:holes_y-1])
      translate([x * aligned_spacing, y * aligned_spacing, -5])
        cylinder(h=15, d=3, $fn=16);
}
// Create pattern in 100x80mm area
grid_mounting_pattern(100, 80, 7.3);
```

Power and Root Functions: pow(), sqrt()

Calculate exponential relationships and distances:

```
// pow(): Raise to power
echo(pow(2, 3)); // 2^3 = 8
echo(pow(10, 2)); // 10^2 = 100
// sqrt(): Square root
echo(sqrt(9)); // 3
echo(sqrt(2)); // 1.41421...
// Practical: Calculate distance between two points
function distance_2d(p1, p2) =
  sqrt(pow(p2[0] - p1[0], 2) + pow(p2[1] - p1[1], 2));
// Distance from [0, 0] to [3, 4] is 5
d = distance_2d([0, 0], [3, 4]);
echo(d); // Prints: 5
```

Practical Example: Parametric Phone Stand Angle Calculation

```
// Calculate stand geometry based on phone height and desired viewing →
    ↪ distance
module calculated_stand(phone_width, phone_height, viewing_distance) {
  // Calculate required back angle for ergonomic viewing
  required_angle = atan(phone_height / viewing_distance);
  // Calculate back support length needed
  base_length = phone_width + 20; // 20mm margin
  back_length = sqrt(pow(base_length, 2) + pow(phone_height + 30, 2));
  echo(str("Calculated angle: ", required_angle, " degrees"));
  echo(str("Back support length: ", back_length, " mm"));
  // Create stand with calculated dimensions
  union() {
    // Base
    cube([base_length, 80, 4]);
    // Back support at calculated angle
    translate([0, -30, 4])
      rotate([required_angle, 0, 0])
        cube([base_length, 10, 4]);
  }
}
```

```
calculated_stand(75, 160, 300); // 75mm phone, 160mm tall, 300mm viewing →
    ↪ distance
```

Vector Math: cross(), dot(), norm()

Advanced positional calculations for complex assemblies:

```
// dot(): Dot product (measure alignment)
// cross(): Cross product (calculate perpendicular)
// norm(): Vector magnitude (length)
// Example: Align two objects perpendicular
function perpendicular_vector(v) =
  cross(v, [0, 0, 1]); // Perpendicular to v in Z plane
// Practical: Create perpendicular support bar
v1 = [1, 0, 0]; // Forward direction
v_perp = perpendicular_vector(v1);
// v_perp is now [0, 1, 0] (sideways)
```

Practical Example: Dynamic Fixture Positioning

```
// Position fixtures based on desired alignment direction
module fixture_on_surface(surface_direction, offset_distance) {
  // Ensure surface_direction is normalized
  normal = surface_direction / sqrt(pow(surface_direction[0], 2) +
                                   pow(surface_direction[1], 2) +
                                   pow(surface_direction[2], 2));
  // Position fixture along surface normal
  translate([normal[0] * offset_distance,
            normal[1] * offset_distance,
            normal[2] * offset_distance])
    cube([10, 10, 10]);
}
// Mount fixture 20mm away from surface
fixture_on_surface([1, 0, 0], 20);
```

Practical Application: Parametric Spiral Design

Here's a complete example combining multiple math functions:

```
// Create a parametric spiral using sin/cos and sqrt
module spiral_generator(max_radius, height, spiral_tightness) {
  steps = 200;
  for (i = [0:steps-1]) {
    // Calculate angle and radius for spiral point
    angle = (i / steps) * spiral_tightness * 360;
    t = i / steps; // Normalized 0-1
    r = max_radius * t; // Radius increases with height
    // Calculate position using trigonometry
    x = r * cos(angle);
    y = r * sin(angle);
    z = (t * height);
    // Place sphere at calculated position
    translate([x, y, z])
      sphere(r=2, $fn=16);
  }
}
spiral_generator(30, 50, 3); // 30mm radius, 50mm tall, 3 complete →
    ↪ revolutions
```

Quiz - Lesson 3dMake.7 (10 questions)

1. What does the `rotate()` function do, and how does it differ from physical rotation[167]?
2. Why is parametric positioning important for design iteration[168]?
3. Explain the Minkowski sum operation and why it's useful for filleting[169].
4. How would you position a second component relative to the first using `translate()` [170]?
5. What parameter would you change to make a phone stand suitable for tablets[171]?
6. True or False: You can rotate an object around multiple axes in a single `rotate()` call.
7. Describe how `$fn` affects the appearance of rounded edges created by Minkowski[172].
8. What advantage does parametric design have over manually modeling each variant[173]?
9. How would you verify that your stand's angle matches your design intent after printing[174]?
10. What design considerations should you account for when adding a lip to prevent phone slippage[175]?

Extension Problems (10)

1. Create five stand variants (for phones, tablets, documents, laptops, and books) by parameterizing width, angle, and lip height[176].
2. Add parametric feet (small cylinders) to the base to improve stability; test with and without feet[177].

[167] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

[168] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

[169] OpenSCAD Manual - Minkowski - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski

[170] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

[171] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[172] OpenSCAD Manual - Minkowski - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski

[173] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

[174] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[175] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[176] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[177] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

3. Use `3dm describe` to document each variant's key geometric properties[178].
4. Design and test a snap-fit connector system that joins the base and back without fasteners[179].
5. Create a comparison table showing print time, material weight, and assembly complexity for each variant[180].
6. Build a complete phone stand product family: define naming convention, parameter ranges, and assembly instructions.
7. Develop a stress analysis guide: identify high-stress areas in your stand and propose reinforcement strategies.
8. Create a customization guide for end users: how to modify angle, width, and lip height for different devices.
9. Design an accessibility-focused stand: include tactile angle markers and clear, non-visual assembly instructions.
10. Write a comprehensive stand design documentation: CAD parameters, material recommendations, print settings, assembly, troubleshooting, and accessible design notes.

Starter Code

Use this parametric phone stand as your starting point:

```
// Parametric Phone Stand - Intermediate Example
// Parameters
thickness = 4; // mm
width = 70; // mm
depth = 90; // mm
angle = 65; // degrees
lipheight = 12; // mm
filletr = 6; // mm (mock fillet by minkowski)
module plate(w, d, t){
  cube([w, d, t], center=false);
}
module fillet(shape, r){
  minkowski(){
    children();
    cylinder(h=0.01, r=r, $fn=40);
  }
}
// Back plate
module back(){
  rotate([angle,0,0])
    fillet(){ plate(width, depth, thickness); }
}
// Base
module base(){
  translate([0,0,0])
    fillet(){ plate(width, depth, thickness); }
}
// Lip
```

[178] OpenSCAD Manual - Transformations - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

[179] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[180] 3DMake GitHub - Phone Stand Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

```
module lip(){
  translate([0, depth-8, thickness])
    cube([width, 8, lipheight], center=false);
}
union(){
  base();
  back();
  lip();
}
```

References

Lesson 8: Advanced Parametric Design and Interlocking Features

Estimated time: 120-150 minutes

Learning Objectives

- Design parametric walls with precise clearances for stackable assemblies[181]
- Create interlocking features (rims, tabs, snap-fits) for assembly without fasteners[182]
- Apply chamfers for edge finishing and print quality[183]
- Manage tolerance stack-up in complex parametric designs[184]

Materials

- 3dMake project scaffold with `src/main.scad`
- Calipers for measuring clearances and wall thickness
- Printer for test prints (recommended for validation)

Related Project: Work through stackable_bins.scad to apply tolerance design in a multi-part assembly with interlocking features.

Understanding Tolerance and Clearance

In 3D printing, tolerance refers to the acceptable variation in printed dimensions. When designing parts that must fit together, you must account for:

- Printer accuracy (~0.1-0.3 mm typical)
- Material shrinkage (PLA: ~0.3-0.5%)
- Cumulative tolerance when multiple parts fit together

[181] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[182] Engineering Tolerance in 3D Printing - https://www.sculpteo.com/en/blog/2021/06/17/tolerance-3d-printing/

[183] OpenSCAD Manual - Minkowski (chamfers/rounding) - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski

[184] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

For stackable bins with interlocking rims:

```
bin_outer = 80 mm
rim_clearance = 0.6 mm (small space between stack and rim below)
rim_inner = bin_outer - 2*wall - 2*rim_clearance
```

Step-by-step Tasks

Task 1: Design a Parametric Stackable Bin

Create a storage bin with walls, a removable lid, and interlocking features:

```
// Stackable Storage Bin - Advanced Parametric Design
// Features: walls, interlocking rim, chamfered edges, optional dividers
// === PARAMETERS ===
bin_w = 80; // mm - width
bin_d = 120; // mm - depth
bin_h = 60; // mm - height
wall = 2; // mm - wall thickness (structural)
rim = 3; // mm - interlock rim height
chamfer = 2; // mm - edge chamfer
stack_clear = 0.6; // mm - clearance between stacking parts
// Derived parameters
inner_w = bin_w - 2*wall;
inner_d = bin_d - 2*wall;
rim_inner_w = bin_w - 2*(wall + stack_clear);
rim_inner_d = bin_d - 2*(wall + stack_clear);
// === MODULES ===
// Outer shell of the bin
module outer_shell() {
  cube([bin_w, bin_d, bin_h]);
}
// Inner cavity (creates walls by subtracting)
module inner_cavity() {
  translate([wall, wall, wall])
    cube([inner_w, inner_d, bin_h - wall]);
}
// Main bin body (hollow box)
module body() {
  difference() {
    outer_shell();
    inner_cavity();
  }
}
// Outer rim (sits on top of bin, inside rim of bin below)
module rim_outer() {
  translate([0, 0, bin_h])
    cube([bin_w, bin_d, rim]);
}
// Inner rim cutout (where the bin below's rim sits)
module rim_inner() {
  translate([wall + stack_clear, wall + stack_clear, bin_h])
    cube([rim_inner_w, rim_inner_d, rim]);
}
// Rim assembly (outer + inner cutout)
module rim_assembly() {
  difference() {
    rim_outer();
    rim_inner();
  }
}
// Chamfer edges for better print quality
module chamfered_edges() {
  difference() {
    children();
    // Top edge chamfer
    translate([-1, -1, bin_h - chamfer])
      cube([bin_w + 2, bin_d + 2, chamfer + 1]);
  }
}
// === ASSEMBLE ===
union() {
  chamfered_edges() {
    body();
  }
  rim_assembly();
}
```

Task 2: Test Stackability

After printing the first bin:

1. Print a second identical bin
2. Stack one on top of the other and verify:
    - Does the rim insert smoothly into the rim below?
    - Can you stack 3 bins without binding?
    - Does the stack remain stable without tipping?

Document:

- Any friction or resistance (good for stability)
- Gaps or looseness (adjust `stack_clear` if needed)

Task 3: Design Variants for Different Storage Needs

Create three different bin sizes using the same parametric code:

```
// Small bin (desk organization)
bin_w = 60;
bin_d = 80;
bin_h = 40;
// OR
// Medium bin (general storage)
// bin_w = 80;
// bin_d = 120;
// bin_h = 60;
// OR
// Large bin (bulk storage)
// bin_w = 120;
// bin_d = 160;
// bin_h = 80;
```

Task 4: Add Optional Internal Dividers

Enhance the design with parametric dividers:

```
// Optional: Parametric dividers
num_dividers = 2; // Number of internal compartments
divider_thickness = 1.5; // mm
module dividers() {
  for (i = [1 : num_dividers]) {
    y_pos = wall + (i * inner_d / (num_dividers + 1));
    translate([wall, y_pos, wall])
      cube([inner_w, divider_thickness, bin_h - 2*wall]);
  }
}
// Add to union:
// dividers();
```

Task 5: Document Tolerance Sensitivity

Create a testing matrix to understand how variations affect fit:

```
Test Configuration Matrix:
========================================
Config | stack_clear | wall | Result
========================================
A | 0.4 mm | 2.0 | Tight - hard to stack
B | 0.6 mm | 2.0 | Ideal - smooth fit
C | 0.8 mm | 2.0 | Loose - unstable
D | 0.6 mm | 2.5 | Better - stronger walls
========================================
```

After printing and testing, record observations about fit quality.

Advanced Topics: Snap-Fit Connectors and Multi-Part Assemblies

As you progress beyond simple stacking, you may want to create assemblies where parts connect without fasteners. This section covers snap-fit and interlocking designs.

Designing Snap-Fit Clips

A snap-fit clip works by:

1. Creating a flexible arm that bends outward to release
2. Providing a catch (usually a small notch or undercut) that locks into place
3. Balancing flexibility (easy to insert/remove) with durability (survives many cycles)

Here's a simple snap-fit connector:

```
// Simple snap-fit clip for connecting parts
module snap_fit_base() {
  // Main mounting block
  cube([20, 40, 15]);
}
module snap_fit_arm(thickness = 1.5, deflection = 0.5) {
  // Flexible arm that bends to insert/release
  // thickness: how thick the arm is (0.8-2.0 mm typical)
  // deflection: how far it must bend to insert (0.3-1.0 mm typical)
  translate([0, 0, 15]) {
    // Thin vertical arm
    cube([thickness, 35, 12]);
    // Catch hook (small bump on end)
    translate([-1, 30, 10])
      cube([3, 5, 2]); // Notch for mating part to lock onto
  }
}
// Test the snap-fit assembly
```

```
union() {
  snap_fit_base();
  translate([3, 0, 0]) snap_fit_arm();
}
```

Key parameters to test:

- `thickness` (1.0-2.0 mm): Thicker = stronger but harder to flex
- Arm length: Longer = more flexible but weaker
- Catch geometry: Size and shape determine how securely parts lock

Tolerance for Snap-Fits

Unlike stackable bins, snap-fit parts need tighter tolerances because they must flex on insertion:

```
Recommended tolerances for snap-fit:
- Arm thickness tolerance: +/-0.2 mm (tighter than bins)
- Catch gap (clearance): 0.3-0.5 mm (smaller than stacking clearance)
- Insertion force: Should be noticeable but not require tools

If too loose: Parts fall apart
If too tight: Parts won't insert or break on insertion
```

Multi-Part Assemblies: Coordinating Multiple Components

For more complex assemblies (like the Miniature Assembly in later projects), you need to:

1. Define a global coordinate system where all parts know their positions
2. Use modular design so each part can be printed, tested, and refined independently
3. Document assembly order so users know which parts fit together first

Here's a pattern for a multi-part assembly:

```
// Multi-part assembly pattern
// === ASSEMBLY PARAMETERS ===
assembly_id = "miniature_furniture"; // Unique name for this assembly
part_scale = 1.0; // Scale entire assembly if needed
// === PARTS ===
module part_base() {
  cube([60, 40, 5]);
}
module part_support_left() {
  translate([5, 0, 5])
    cube([3, 40, 25]);
}
module part_support_right() {
  translate([52, 0, 5])
    cube([3, 40, 25]);
}
module part_shelf() {
  translate([8, 5, 18])
    cube([44, 30, 2]);
}
// === ASSEMBLY ===
// Complete assembly (print individual parts separately first)
module furniture_assembly() {
  part_base();
  part_support_left();
  part_support_right();
  part_shelf();
}
// Export individual parts by uncommenting one at a time:
// part_base();
// part_support_left();
// part_support_right();
// part_shelf();
// Or export the full assembly:
furniture_assembly();
```

Best practices for multi-part designs:

1. Print and test parts individually before assembling
2. Add small alignment features (pins, notches) to guide assembly
3. Document which parts go together and the order
4. Include spare parts (print 2-3 copies of small connecting pieces)
5. Test the full assembly on a test print before final production

Assembly Documentation

For any multi-part project, create a simple assembly guide:

```
Miniature Furniture Assembly Guide

Parts:
- Base (60 x 40 x 5 mm, prints in ~20 min)
- Support Left (3 x 40 x 25 mm, prints in ~5 min)
- Support Right (3 x 40 x 25 mm, prints in ~5 min)
- Shelf (44 x 30 x 2 mm, prints in ~15 min)

Assembly Steps:
1. Prepare all four parts and test fit manually
2. Glue or snap supports to base (supports on left and right edges)
3. Slide shelf onto supports
4. Check that shelf is level and square

Quality Checks:
- Shelf should not wobble or flex excessively
- All parts should be free of support marks or defects
- Shelf should hold light objects without sagging
```

Advanced Assembly Best Practices

Professional assemblies require planning beyond individual parts. Consider assembly sequence, tolerance management, and accessibility.

Designing for Snap-Fit Assemblies

Snap-fits eliminate fasteners, reducing assembly complexity and cost:

```
// Snap-fit connector with spring arm
module snap_fit_male(width, depth, arm_thickness, flex_distance) {
  difference() {
    // Main body
    cube([width, depth, 10]);
    // Undercut for spring arm
    translate([flex_distance, 2, 2])
      cube([width - 2*flex_distance, depth - 4, 6]);
  }
  // Spring arm that flexes inward
  translate([flex_distance, 2, 2])
    cube([arm_thickness, depth - 4, 6]);
}
// Snap-fit receiver with clip pockets
module snap_fit_female(width, depth) {
  cube([width, depth, 10]);
  // Clip pockets on inside
  translate([3, 2, 5])
    cube([width - 6, depth - 4, 4]);
}
// Usage: Position parts for assembly
snap_fit_male(40, 30, 2, 35);
translate([50, 0, 0])
  snap_fit_female(40, 30);
```

Complete Snap-Fit Housing Example:

```
// Parametric connector box with snap-fit lid
module connector_housing(box_w, box_h, box_d, wall, snap_depth) {
  difference() {
    // Outer shell
    cube([box_w, box_h, box_d]);
    // Hollow interior
    translate([wall, wall, wall])
      cube([box_w - 2*wall, box_h - 2*wall, box_d - wall]);
  }
  // Snap-fit ledges on sides
  for (z = [box_d - 5, box_d - 2])
    for (x = [0, box_w - wall])
      translate([x, 0, z])
        cube([wall, box_h, 3]);
}
module connector_lid(box_w, box_h, snap_depth, tab_thickness) {
  difference() {
    cube([box_w, box_h, 2]);
    // Hollow underside
    translate([tab_thickness, tab_thickness, 0])
      cube([box_w - 2*tab_thickness, box_h - 2*tab_thickness, 2]);
  }
  // Snap tabs underneath
  for (x = [0, box_w - tab_thickness])
    translate([x, tab_thickness, -2])
      cube([tab_thickness, box_h - 2*tab_thickness, 2]);
}
// Assembly visualization
connector_housing(60, 40, 30, 2, 3);
translate([0, 0, 30])
  connector_lid(60, 40, 3, 3);
```

Thread and Screw Calculations

Design for M3 and M4 fasteners commonly available in 3D printing accessories:

```
// M3 bolt hole specifications
M3_clearance_hole = 3.2; // Loose fit, easy insertion
M3_tight_hole = 2.8; // Tight fit for threaded inserts
M3_head_diameter = 5.5; // M3 bolt head diameter
M3_head_height = 2.8; // M3 bolt head height
// M4 bolt hole specifications
M4_clearance_hole = 4.3;
M4_tight_hole = 3.8;
M4_head_diameter = 7;
```

```
M4_head_height = 3.2;
// Threaded insert pocket (wooden threaded inserts for plastic)
module threaded_insert_pocket(bolt_type, pocket_depth) {
  hole_diameter = bolt_type == "M3" ? 3.5 : 4.5; // Insert body
  head_diameter = bolt_type == "M3" ? 5.8 : 7.5; // Install head
  difference() {
    cylinder(r=head_diameter/2, h=pocket_depth);
    cylinder(r=hole_diameter/2, h=pocket_depth + 2);
  }
}
// Mounting bracket with threaded insert pockets
module mounting_bracket(length, height, pocket_depth) {
  difference() {
    cube([length, 20, height]);
    // Create pockets for threaded inserts
    for (x = [10, length - 10])
      translate([x, 10, height - pocket_depth])
        threaded_insert_pocket("M3", pocket_depth);
  }
}
mounting_bracket(60, 20, 5);
```

Alignment Features and Dowel Pins

Guide parts into correct assembly positions:

```
// Dowel pin socket (receives alignment pin)
module dowel_socket(diameter, depth) {
  difference() {
    cylinder(r=diameter/2 + 2, h=depth);
    cylinder(r=diameter/2 + 0.1, h=depth);
  }
}
// Dowel pin (protrudes from part)
module dowel_pin(diameter, height) {
  cylinder(r=diameter/2, h=height);
}
// Part A with dowel pins
module part_a() {
  difference() {
    cube([50, 50, 10]);
    translate([10, 10, -2])
      cylinder(r=2, h=5); // Nut pocket
  }
  // Alignment pins on top
  translate([10, 10, 10])
    dowel_pin(4, 5);
  translate([40, 10, 10])
    dowel_pin(4, 5);
}
// Part B with dowel sockets
module part_b() {
  difference() {
    cube([50, 50, 10]);
    // Socket for alignment pins from Part A
    translate([10, 10, 10])
      dowel_socket(4, 5);
    translate([40, 10, 10])
      dowel_socket(4, 5);
  }
}
// Assembly visualization
part_a();
translate([0, 0, 15]) color("blue", 0.5) part_b();
```

Assembly Sequence Documentation

Structure complex assemblies with clear ordering:

```
// Multi-stage assembly documented in code
module complete_assembly(config_name) {
  echo(str("Assembling: ", config_name));
  // STAGE 1: Place base components
  echo("STAGE 1: Install base bracket");
  base_bracket();
  // STAGE 2: Add vertical structure
  echo("STAGE 2: Attach vertical supports");
  translate([0, 0, 20])
    vertical_support();
  // STAGE 3: Install cross members
  echo("STAGE 3: Connect cross members");
  translate([20, 0, 30])
    cross_member();
  // STAGE 4: Final fastening
  echo("STAGE 4: Secure fasteners (M3 bolts)");
  // Fasteners would be positioned here
}
complete_assembly("standard_shelf");
```

Exploded View Generation

Create exploded assembly drawings for documentation:

```
// Create exploded view by offsetting parts
module exploded_assembly() {
  explode_factor = 30; // mm spacing between parts
  // Base (stays at origin)
  color("red")
    base_component();
  // Second component (moved up and forward)
  translate([0, explode_factor, 0])
    color("green")
      middle_component();
  // Third component (moved further)
  translate([0, explode_factor*2, 0])
    color("blue")
      top_component();
  // Fasteners (moved separately)
  for (i = [0:3])
    translate([i*10, explode_factor*2, 20])
      color("gray")
        cylinder(r=2, h=10);
}
exploded_assembly();
```

Testing Interference and Clearance

```
// Visualization: show clearance zones (semi-transparent)
module assembly_with_clearance_check() {
  // Actual parts
  part_1();
  translate([30, 0, 0]) part_2();
  // Clearance zones (show minimum spacing)
  % translate([25, 0, 0]) // % makes transparent
    cube([1, 50, 50]); // 1mm clearance zone
}
// When rendered, the % shows potential interference areas
```

Checkpoint

Chamfers improve print quality by:

1. Reducing stress concentration at edges
2. Improving layer adhesion at sharp transitions
3. Making parts easier to remove from the build platform

```
// More sophisticated chamfer (chamfered corner)
module chamfer_corner(size, radius) {
  difference() {
    children();
    translate([size-radius, size-radius, size-radius])
      rotate([45, 0, 0]) cube([radius*2, radius*2, radius*2]);
  }
}
```

Checkpoint

- After task 1, you have a single parametric bin
- After task 2, you've verified stackability
- After task 4, you have optional dividers working
- After task 5, you've documented tolerance data

Quiz - Lesson 3dMake.8 (10 questions)

1. What is the purpose of the `stack_clear` parameter in stackable bin design[185]?
2. Why is wall thickness a critical parameter in structural parts[186]?
3. Explain what ”tolerance stack-up” means and why it matters in multi-part assemblies[187].
4. How does chamfering improve print quality[188]?
5. What parameter would you adjust to make bins stack with more friction for stability[189]?
6. True or False: Thin walls always print faster than thick walls.
7. Describe the difference between the outer rim and inner rim in the stackable bin design[190].

[185] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md
[186] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md
[187] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md
[188] OpenSCAD Manual - Minkowski (chamfers/rounding) - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski
[189] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md
[190] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

8. Why might you use `num_dividers` as a parameter instead of hard-coding divider positions[191]?
9. How would you measure whether your printed bin matches the designed dimensions[192]?
10. What design considerations apply when stacking more than 3 bins vertically[193]?

Extension Problems (10)

1. Create five bin variants (small, medium, large, tall, shallow) using parametric logic; print and stack them[194].
2. Add parametric rounded corners using `minkowski()` to improve structural integrity[195].
3. Design a custom divider system: parameterize number, position, and thickness; test strength[196].
4. Conduct a tolerance study: print bins with `stack_clear` values of 0.4, 0.6, 0.8 mm; document fit quality[197].
5. Create a modular stacking system with connecting clips or latches; test assembly and disassembly[198].
6. Build a complete storage system: design multiple bin sizes with standard stackability and create an assembly guide.
7. Develop a manufacturing specification document: tolerance ranges, material recommendations, quality acceptance criteria.
8. Design a labeling system: add parametric label holders or embossed areas for identifying bin contents.
9. Create a structural analysis guide: identify stress concentration points and propose reinforcement strategies.
10. Write comprehensive bin design documentation: parameters, variants, assembly, stackability testing, tolerance management, maintenance, and accessibility features.

References

[191] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[192] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[193] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[194] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[195] OpenSCAD Manual - Minkowski (chamfers/rounding) - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski

[196] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[197] 3DMake GitHub - Stackable Bins Example - https://github.com/tdeck/3dmake/blob/main/docs/examples.md

[198] Engineering Tolerance in 3D Printing - https://www.sculpteo.com/en/blog/2021/06/17/tolerance-3d-printing/

Snap-Fit Clip Project

Starter Code

Use this stackable bins example as your starting point:

```
// Stackable Storage Bins - Advanced Example
// Parameters
binw = 80; // width
bind = 120; // depth
binh = 60; // height
wall = 2; // wall thickness
rim = 3; // interlock rim height
chamfer = 2;
stackclear = 0.6; // clearance between stacks
module outer(){
  cube([binw, bind, binh]);
}
module inner(){
  translate([wall, wall, wall])
    cube([binw-2*wall, bind-2*wall, binh-2*wall]);
}
module body(){
  difference(){ outer(); inner(); }
}
module rimouter(){
  translate([0,0,binh]) cube([binw, bind, rim]);
}
module riminner(){
  translate([wall+stackclear, wall+stackclear, binh])
    cube([binw-2*(wall+stackclear), bind-2*(wall+stackclear), rim]);
}
module chamferedges(){
  // Simple top edge chamfer via difference
  difference(){
    children();
    translate([-1,-1,binh-chamfer]) cube([binw+2, bind+2, chamfer+2]);
  }
}
union(){
  chamferedges(){ body(); }
  rimouter();
  difference(){ rimouter(); riminner(); }
}
```

Snap-Fit Clip - Student Documentation Template (Extension Project)

- Author:
- Date:
- Description: Design a snap-fit connector that joins two 3D-printed parts without external fasteners.

Design Concept

- What two parts will your snap-fit connect?
- Design approach (cantilever, torsion box, other):
- Material and expected stress limits:

Tolerance Specifications

Parameter	Nominal Value	Tolerance	Rationale
Tab thickness			
Clip opening			
Undercut depth			
Contact area			

Assembly Testing

Test 1: Initial Assembly

- Date:
- Effort required to snap together:
- Snap strength (resistance to pulling apart):
- Visual inspection (any deformation?):

Test 2-5: Reusability Testing

- Cycle 2: (date, observations)
- Cycle 3: (date, observations)
- Cycle 4: (date, observations)
- Cycle 5: (date, observations)

Durability Assessment

- After 5 cycles, is the snap-fit still functional?
- Any permanent deformation observed?
- Estimated service life (based on testing):

Mechanical Analysis

- Describe the snap mechanism (why does it work?):
- What material properties enable this design?
- What are the stress points?

Reflections

- Did the snap-fit work as expected?
- What tolerance adjustments would you make?
- How would you test durability over thousands of cycles?
- What materials would fail with this design? Why?

Attachments

- ☐ `.scad` file with parametric snap-fit module
- ☐ Photos of both parts before assembly
- ☐ Photos of assembled connection
- ☐ Testing documentation with dates/observations
- ☐ Tolerance analysis table

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Snap-Fit Clip - Teacher Template (Extension Project)

Briefing

Students design a snap-fit connector that joins two 3D-printed parts without fasteners. This project emphasizes tolerance engineering, material properties, and mechanical design validation.

Key Learning: Tolerance engineering; snap-fit mechanics; non-destructive assembly testing.

Real-world Connection: Snap-fits are ubiquitous in consumer products. Designing them requires understanding material stress, geometry precision, and repeated-use durability.

Constraints

- Snap-fit must connect two printed parts without external fasteners
- Design must be parametric (tolerance variables clearly labeled)
- Assembly must be testable (connection strength, reusability)
- Design must account for material properties (PLA creep, flex limits)

Functional Requirements

- Parts snap together securely without over-constraint
- Connection withstands intended use without permanent deformation
- Snap mechanism is reusable (at least 5+ assembly cycles)
- Design demonstrates understanding of tolerance and material behavior

Deliverables

- `.scad` with parametric snap-fit module
- Completed documentation template
- Assembly testing log (snap strength, cycle durability)
- Tolerance analysis documentation
- Reflection on mechanical design trade-offs

Rubric

Category 1: Problem & Solution (0-3) Snap-fit works reliably; parts assemble and disassemble as designed.

Category 2: Design & Code Quality (0-3) Code shows tolerance thinking. Design is mechanically sound.

Category 3: Documentation (0-3) Assembly testing documented. Tolerance rationale explained.

Assessment Notes

- Strong submissions: Show evidence of tolerance testing, multiple assembly cycles documented, and reflection on material limits
- Reinforce: Snap-fit design principles; tolerancing for manufacturability
- Extension: Material property analysis; cost-benefit comparisons

Lesson 9: Automation and 3dm Workflows

Estimated time: 90-120 minutes

Learning Objectives

- Chain 3dm commands to automate the design-to-print pipeline[199]
- Create shell scripts that manage multiple model variants automatically[200]
- Use `3dm lib` for library management and code reuse[201]
- Optimize workflows for batch processing and continuous improvement[202]

Materials

- 3dMake project scaffold with multiple model files

[199] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[200] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[201] 3DMake Library Management - https://github.com/tdeck/3dmake/wiki/Library-Management
[202] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

- Terminal/command line access
- Text editor for script creation

The 3dm Command Chain

Instead of running commands one at a time, you can chain them to automate the entire workflow:

```
# Individual commands (slow, manual)
3dm build src/main.scad
3dm describe src/main.scad
3dm slice src/main.scad

# Chained commands (fast, automatic)
3dm build src/main.scad && 3dm describe src/main.scad && 3dm slice src/main.
    ↪ scad
```

The && operator ensures each command runs only if the previous one succeeds.

Step-by-step Tasks

Task 1: Create a Multi-Variant Project

Organize your project with multiple model files:

```
src/
+------ main.scad # Base model
+------ keycap_small.scad # Variant 1
+------ keycap_medium.scad # Variant 2
+------ keycap_large.scad # Variant 3
+------ stand_test.scad # Experimental model
```

Each file should be independent and buildable:

```
// keycap_small.scad
key_size = 14;
key_height = 10;
letter = "S";
// ... rest of keycap code
```

Task 2: Create a Batch Build Script

Create `build_all.sh` to build, describe, and slice all models:

```
#!/bin/bash
# build_all.sh - Batch process all SCAD files

echo "=== Building all models ==="

for scad_file in src/*.scad; do
  echo "Processing: $scad_file"

  # Extract filename without extension
  base_name=$(basename "$scad_file" .scad)

  # Build
  echo " Building..."
  3dm build "$scad_file" || continue
  # Describe (AI analysis if configured)
  echo " Describing..."
  3dm describe "$scad_file" > "build/${base_name}.description.txt"

  # Slice (generate G-code)
  echo " Slicing..."
  3dm slice "$scad_file"

  # Move G-code to organized folder
  if [ -f "build/main.gcode" ]; then
    mv "build/main.gcode" "build/${base_name}.gcode"
  fi

  echo " Done: $base_name"
done

echo "=== All models processed ==="
ls -lh build/*.gcode
```

To run the script:

```
chmod +x build_all.sh # Make executable
./build_all.sh # Run
```

Task 3: Create a Variant Testing Script

Test parameter variations automatically and compare results:

```
#!/bin/bash
# test_variants.sh - Build and compare parameter variants

echo "=== Testing keycap variants ==="

# Array of sizes to test
sizes=(12 16 20 24)

for size in "${sizes[@]}"; do
  echo "Testing size: ${size}mm"

  # Create temporary SCAD with this size
  cat > temp_keycap.scad << EOF
key_size = $size;
key_height = 12;
wall = 1.2;
letter = "T";

module shell(){
  difference(){
    cube([key_size, key_size, key_height], center=false);
    translate([wall, wall, wall])
      cube([key_size-2*wall, key_size-2*wall, key_height], center=false);
  }
}

module emboss(){
  translate([key_size/2, key_size/2, key_height-0.01])
    linear_extrude(height=0.8)
      text("T", size=10, halign="center", valign="center");
}

union(){
  shell();
  emboss();
}
EOF

  # Build and analyze
  3dm build temp_keycap.scad
  # Get file size (proxy for material volume)
  stl_file="build/temp_keycap.stl"
  file_size=$(stat -f%z "$stl_file" 2>/dev/null || stat -c%s "$stl_file" 2>/ →
      ↪ dev/null)

  echo " -> STL size: $(numfmt --to=iec-i --suffix=B "$file_size" 2>/dev/ →
      ↪ null || echo "$file_size bytes")"

  # Save for comparison
  cp "$stl_file" "build/keycap_${size}mm.stl"
done

echo "=== Variant testing complete ==="
echo "Results saved to build/"
```

Task 4: Library Management

Organize reusable modules into libraries:

```
# Create a library structure
mkdir -p lib
mkdir -p lib/connectors
mkdir -p lib/shapes
mkdir -p lib/fasteners

# Create lib/shapes/rounded_cube.scad
cat > lib/shapes/rounded_cube.scad << 'EOF'
// Rounded cube module for reuse across projects

module rounded_cube(w, h, d, radius, fn=16) {
  minkowski(){
    cube([w-2*radius, h-2*radius, d-2*radius], center=true);
    cylinder(r=radius, h=0.01, $fn=fn);
  }
}

// Example usage:
// rounded_cube(20, 20, 20, 2);
EOF

# View available libraries
3dm lib list

# Install a community library
3dm lib install BOSL2
```

Then in your main.scad:

```
include <lib/shapes/rounded_cube.scad>
// Use the custom module
rounded_cube(30, 20, 15, 2);
```

Task 5: Continuous Integration Concept

Create a workflow that regenerates designs when parameters change:

```
#!/bin/bash
# watch_and_build.sh - Rebuild whenever SCAD files change

watch_dir="src"
build_dir="build"

echo "Watching for changes in $watch_dir..."
```

```
while true; do
  # Find recently modified files
  find "$watch_dir" -name "*.scad" -mmin -1 | while read file; do
    echo "$(date '+%Y-%m-%d %H:%M:%S') - Change detected: $file"

    # Rebuild
    3dm build "$file" && echo " Build successful"
  done

  sleep 5 # Check every 5 seconds
done
```

Batch Processing Workflow

Complete pipeline from code to print-ready files:

```
#!/bin/bash
# production_build.sh - Complete pipeline

PROJECT="my_keycaps"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
OUTPUT_DIR="builds/$PROJECT/$TIMESTAMP"

mkdir -p "$OUTPUT_DIR"

echo "=== Production Build: $PROJECT ($TIMESTAMP) ==="

# Step 1: Build all models
for scad in src/*.scad; do
  name=$(basename "$scad" .scad)
  echo "Building: $name"
  3dm build "$scad" || { echo "Failed: $name"; exit 1; }
  cp "build/main.stl" "$OUTPUT_DIR/${name}.stl"
  cp "build/main.gcode" "$OUTPUT_DIR/${name}.gcode"
done

# Step 2: Generate summaries
echo "Generating build report..."
{
  echo "# Production Build Report"
  echo "Date: $(date)"
  echo "Project: $PROJECT"
  echo ""
  echo "## Files Generated"
  ls -lh "$OUTPUT_DIR"
  echo ""
  echo "## Total Material Required"
  du -sh "$OUTPUT_DIR"
} > "$OUTPUT_DIR/BUILD_REPORT.md"

# Step 3: Archive
echo "Archiving..."
tar -czf "${OUTPUT_DIR}.tar.gz" "$OUTPUT_DIR"

echo "=== Build complete ==="
echo "Output: $OUTPUT_DIR"
echo "Archive: ${OUTPUT_DIR}.tar.gz"
```

File Import/Export: Working with External Assets

Professional OpenSCAD projects often incorporate external files-DXF drawings, SVG logos, or shared library modules. Understanding import, include, and use is critical for managing complex projects.

import(): Loading 3D Files (DXF, STL, AMF)

Import 3D geometry from external files. Useful for incorporating CAD data or third-party parts:

```
// Import DXF profile and extrude it
import("path/to/profile.dxf"); // Load 2D or 3D geometry
// Practical example: Import CAD drawing and extrude
module extruded_part() {
  linear_extrude(height=10)
    import("technical_drawing.dxf");
}
// Scale imported geometry (OpenSCAD units may differ from source)
scale([1, 1, 10]) // Scale Z by 10 if DXF uses different units
  import("part.dxf");
```

Practical Example: Combining CAD and Parametric Design

```
// Import a technical DXF sketch and create 3D model around it
module custom_bracket() {
  // Import 2D profile from technical drawing
  difference() {
    linear_extrude(height=5)
      scale(25.4) // Convert from inches to mm
        import("bracket_profile.dxf");
    // Add mounting holes (parametric)
    for (x = [10, 40])
      for (y = [10, 40])
        translate([x, y, -5])
          cylinder(h=15, d=3, $fn=16);
  }
}
custom_bracket();
```

include vs use: Library Management

Both load external SCAD files, but with different scoping:

```
// include: Imports functions and modules into current namespace
// Use this when you want all definitions merged into current file
include <library.scad>
// Now you can use any function/module defined in library.scad
my_shape();
// ==========================================
// use: Imports but creates separate namespace
// Use this when you want to avoid name conflicts
use <library.scad>
// You must prefix functions/modules with filename (no extension)
library:my_shape();
```

Practical Example: Multi-File Project Structure

```
// main.scad - Main project file
// Load reusable components
include <utilities.scad> // Common helpers
include <fasteners.scad> // Bolt/screw definitions
include <connectors.scad> // Joining mechanisms
// Define local modules
module housing() {
  difference() {
    cube([100, 100, 50]);
    translate([10, 10, 10])
      cube([80, 80, 40]);
  }
}
// Use imported components
union() {
  housing();
  // Position imported fasteners
  translate([10, 10, 50])
    bolt_m3(length=10);
  translate([90, 10, 50])
    bolt_m3(length=10);
}
```

Content of utilities.scad (included file):

```
// utilities.scad - Reusable helper functions
// Create repeating pattern
module repeat_linear(count, spacing) {
  for (i = [0:count-1])
    translate([i * spacing, 0, 0])
      children();
}
// Create grid of components
module grid_pattern(cols, rows, spacing) {
  for (x = [0:cols-1])
    for (y = [0:rows-1])
      translate([x * spacing, y * spacing, 0])
        children();
}
function calculate_bolt_clearance(bolt_diameter) = bolt_diameter + 0.5;
```

Export Formats: DXF, SVG, AMF

After creating your model, export it in various formats:

```
# Render to STL (standard 3D printing format)
3dm slice src/main.scad

# Export specific formats using OpenSCAD command line
openscad -o output.dxf src/main.scad # 2D projection as DXF
openscad -o output.svg src/main.scad # 2D projection as SVG
openscad -o output.amf src/main.scad # AMF (supports color/material)
```

Practical Example: Complete Export Workflow

```
#!/bin/bash
# export_variants.sh - Export design in multiple formats

INPUT_FILE="src/main.scad"
OUTPUT_DIR="build/exports"

mkdir -p "$OUTPUT_DIR"

echo "Exporting $INPUT_FILE in multiple formats..."

# STL (3D printing)
openscad -o "$OUTPUT_DIR/model.stl" "$INPUT_FILE"
echo "[YES] STL export complete"

# DXF (CAD compatible)
openscad -o "$OUTPUT_DIR/model_projection.dxf" "$INPUT_FILE"
echo "[YES] DXF export complete"

# SVG (web and vector graphics)
openscad -o "$OUTPUT_DIR/model_projection.svg" "$INPUT_FILE"
echo "[YES] SVG export complete"
```

```
# Generate report
echo ""
echo "=== Export Summary ==="
ls -lh "$OUTPUT_DIR"
```

Advanced: Dynamic File Loading Based on Parameters

Create designs that load different components based on configuration:

```
// Dynamic library loader
// Automatically selects component based on parameter
material = "aluminum"; // or "steel", "plastic"
component_type = "bracket"; // or "connector", "fastener"
// Load appropriate component library
include <materials/aluminum_properties.scad>
include <components/bracket_library.scad>
include <components/connector_library.scad>
// Create part using selected material properties
module material_part(dimensions) {
  material_density = material_density_aluminum; // From included file
  material_strength = material_strength_aluminum;
  // Apply material-specific design rules
  wall_thickness = (material_strength < 50) ? 3 : 2;
  difference() {
    cube(dimensions);
    translate([2, 2, 2])
      cube([dimensions[0]-4, dimensions[1]-4, dimensions[2]-4]);
  }
}
material_part([50, 50, 30]);
```

File Path Handling and Organization

Best practices for organizing files in larger projects:

```
// Recommended project structure
/*
my_project/
+------ src/
| +------ main.scad
| +------ components/
| | +------ fasteners.scad
| | +------ connectors.scad
| | +------ mounting.scad
| +------ libraries/
| +------ utilities.scad
| +------ math_functions.scad
+------ assets/
| +------ sketches/
| | +------ bracket_profile.dxf
| | +------ panel_layout.svg
| +------ references/
| +------ part_specifications.pdf
+------ build/
    +------ main.stl
    +------ exports/
    | +------ main.dxf
    | +------ main.svg
    +------ documentation/
*/
// In main.scad, use relative paths:
include <components/fasteners.scad>
include <components/connectors.scad>
// For assets in different directory:
```

```
module panel_with_logo() {
  difference() {
    cube([100, 100, 5]);
    // Import logo if available
    translate([50, 50, 4.5])
      linear_extrude(0.8)
        import("../assets/sketches/logo.svg");
  }
}
```

Complete Multi-File Assembly Example

```
// main.scad - Complete assembly with imports and exports
include <components/housing.scad>
include <components/internal_support.scad>
use <utilities/positioning_helpers.scad>
// Main assembly
module complete_product() {
  // Housing (from included file)
  housing_shell();
  // Internal support structures
  color("lightgray")
    internal_reinforcement();
  // Add imported external parts
  translate([0, 0, 20])
    import("../assets/pcb_holder.dxf");
}
// Render the complete product
complete_product();
// Export notes:
// - STL: Full 3D model for 3D printing
// - DXF: 2D projection for laser cutting or CNC
// - SVG: 2D artwork for documentation
// - AMF: Multi-material format (if colors matter)
```

Checkpoint

- After task 2, you have a working batch build script
- After task 3, you've tested multiple parameter variants
- After task 5, you understand continuous integration concepts

Quiz - Lesson 3dMake.9 (10 questions)

1. What does the `&&` operator do in a command chain[203]?
2. Why would you use a batch build script instead of running commands manually[204]?
3. How would you extract the filename without extension in a bash script[205]?
4. Explain the purpose of creating a temporary SCAD file for variant testing[206].

[203] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[204] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[205] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[206] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/a

5. What is `3dm lib list` used for[207]?
6. True or False: You must manually edit each model file to test different parameters.
7. Describe what a continuous integration workflow accomplishes[208].
8. How would you save variant test results for comparison[209]?
9. What file format would you use to archive completed builds for long-term storage[210]?
10. How could you use scripting to automatically generate documentation for your designs[211]?

Extension Problems (10)

1. Create a batch build script that processes 5+ model variants and generates a comparison report[212].
2. Design a parameter testing matrix: test 3 dimensions x 3 wall thicknesses and compare results[213].
3. Build a library module for common features (brackets, connectors, fasteners); use in multiple projects[214].
4. Create a ”watch and rebuild” script that automatically regenerates designs when code changes[215].
5. Develop a production workflow script that generates timestamped builds with complete documentation[216].
6. Build an automated testing framework: validate designs against dimensional requirements[217].
7. Create a version control integration: automatically tag and archive successful builds[218].

utomation.md

[207] 3DMake Library Management - https://github.com/tdeck/3dmake/wiki/Library-Management

[208] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[209] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[210] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[211] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[212] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[213] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[214] 3DMake Library Management - https://github.com/tdeck/3dmake/wiki/Library-Management

[215] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[216] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[217] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

[218] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md

8. Design a design parameter database: store and query historical variants and their properties[219].
9. Develop a remote print queue system: batch multiple models and send to printer automatically[220].
10. Write a comprehensive automation documentation: best practices, script templates, troubleshooting, and accessibility considerations.

Helpful Shell Commands Reference

Command	Purpose
`chmod +x script.sh`	Make script executable
`./script.sh`	Run a shell script
`for file in *.scad; do ... done`	Loop through files
`basename file.scad .scad`	Remove extension
`stat -c%s file` (Linux) or `stat -f%z file` (macOS)	Get file size
`find dir -name "*.scad"`	Find all SCAD files
`tar -czf archive.tar.gz folder/`	Create compressed archive
`du -sh folder/`	Show folder size

References

Lesson 10: Hands-On Practice Exercises and Troubleshooting

Estimated time: 120-150 minutes (multiple activities)

Learning Objectives

- Apply all prior lessons in integrated projects[221]
- Diagnose and fix common 3D printing and OpenSCAD issues[222]
- Validate models using deterministic and AI-assisted inspection[223]
- Document design decisions and troubleshooting processes[224]

Materials

[219] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[220] 3DMake GitHub - Automation Examples - https://github.com/tdeck/3dmake/blob/main/docs/automation.md
[221] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[222] 3D Printing Troubleshooting Guide - https://www.prusa3d.com/support/
[223] 3DMake Documentation - https://github.com/tdeck/3dmake/wiki/Troubleshooting
[224] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

- 3dMake project scaffold
- Printer for test prints (recommended)
- Measuring tools (calipers, protractor)
- Access to 3dm commands
- Reference: master-rubric.md for assessment criteria
- Reference: filament-comparison-table.md for material properties

Extension Projects: Complete Dice_Dice_Dice or Snap_Fit_Clip to practice integrated validation workflows.

Measurement Fundamentals

Before you can validate that your designs print correctly, you need to measure accurately. This section covers using calipers and interpreting measurement data.

Understanding Digital Calipers

A digital caliper is a precision tool with three measurement modes:

Mode	Use	Example
Outside jaws	Measure outer dimensions	Diameter of a cylinder, width of a part
Inside jaws	Measure inner dimensions	Hole diameter, inside width of a box
Depth rod	Measure depth or thickness	Depth of a recess, thickness of a part

Measurement Best Practices

1. Zero the caliper: With jaws closed, press the ON/ZERO button to confirm it reads 0.0 mm
2. Use gentle pressure: Tighten jaws just enough to hold the object; excessive force causes false readings
3. Take three trials: Measure the same feature three times and calculate the average
4. Record to 0.1 mm: Most printed parts can be reliably measured to one decimal place

Example measurement sequence:

```
Object: 3D-printed cube
Trial 1: 24.5 mm
Trial 2: 24.2 mm
Trial 3: 24.6 mm
Average: (24.5 + 24.2 + 24.6) / 3 = 24.43 mm 24.4 mm

Interpretation: Designed as 25.0 mm, printed as 24.4 mm
Deviation: -0.6 mm (0.24% shrinkage, acceptable for PLA)
```

Using Measurements to Improve Design

When you find a deviation between your designed and actual dimensions, you can adjust future prints:

```
// Design parameter
part_height = 50; // Designed: 50 mm
// After first print, measured: 49.7 mm
// Shrinkage: ~0.6% (typical for PLA)
// Adjustment for next print:
// Option 1: Add 0.3 mm to compensate
part_height = 50.3; // Should now print ~50.0 mm
// OR
// Option 2: Scale the entire design by 1.006 (0.6% larger)
scale([1.006, 1.006, 1.006])
  main_model();
```

Tolerance Stack-Up in Assemblies

When multiple parts fit together, measurement errors compound. This is called tolerance stack-up:

```
Example: Stackable bins
Designed: Outer width = 80.0 mm, Rim clearance = 0.6 mm

If each part prints 0.3 mm undersized:
Bin A actual: 79.7 mm outer
Bin B actual: 79.7 mm outer

Stack-up effect: Instead of 0.6 mm clearance, you get 0.9 mm
Result: Bins stack loosely (might be acceptable or problematic depending on
    use)

Solution: Design with 0.8 mm clearance initially, measure test prints,
then adjust if needed.
```

Creating a Measurement Log

For any parametric design project, maintain a measurement log:

```
Project: Phone Stand Variants
Date: February 10, 2026

Part: Base (narrow variant, 60 mm)
Design dimension: 60.0 mm width
Trial 1: 59.7 mm
Trial 2: 59.8 mm
Trial 3: 59.9 mm
Average: 59.8 mm
Deviation: -0.2 mm
Status: Acceptable (within +/-0.5 mm tolerance)

Part: Base (wide variant, 120 mm)
Design dimension: 120.0 mm width
Trial 1: 119.5 mm
Trial 2: 119.6 mm
Trial 3: 119.8 mm
Average: 119.6 mm
Deviation: -0.4 mm
Status: Acceptable

Part: Lip (lip_height parameter: 15 mm)
Design dimension: 15.0 mm
Trial 1: 14.8 mm
Trial 2: 14.9 mm
Trial 3: 14.9 mm
Average: 14.87 mm 14.9 mm
Deviation: -0.1 mm
Status: Excellent
```

Reference: Appendix C

For comprehensive QA procedures, assembly testing, and durability validation, see Appendix C: Tolerance Testing & QA Matrix, which covers:

- Go/no-go gauges for functional validation
- Multi-cycle durability testing (snap-fits, hinges)
- Stress testing procedures
- Complete measurement worksheets

Exercise Set A: Guided Projects with Real-World Constraints

Exercise A1: The Parametric Phone Stand (Beginner)

Objective: Create a phone stand that works for 3+ phone models with different screen sizes.

Requirements:

- Base must support 300g weight
- Angle must be adjustable (45-75)
- Width must accommodate phones 60-90mm wide
- All parameters at top of file

Starter Code:

```
// Parametric Phone Stand v1
// Required parameters
phone_width = 75; // mm - adjust for different phones
base_width = phone_width + 10; // Add margin
base_depth = 100; // mm
base_thickness = 5; // mm
stand_angle = 60; // degrees - adjust viewing angle
lip_height = 15; // mm - hold phone at top
// Module definitions
module base() {
  cube([base_width, base_depth, base_thickness]);
}
module stand() {
  rotate([stand_angle, 0, 0])
    cube([base_width, base_depth, base_thickness]);
}
module lip() {
  translate([0, base_depth - 8, base_thickness])
    cube([base_width, 8, lip_height]);
}
// Assemble
union() {
  base();
  stand();
  lip();
}
```

Tasks:

1. Build the model and verify it's manifold with `3dm describe`
2. Generate 3 variants: iPhone (60mm), iPad mini (100mm), Tablet (150mm)
3. Test each variant by measuring critical dimensions
4. Document which variant works best with real devices

Success Criteria:

- All 3 variants build without errors
- Printed parts successfully hold test phones
- Dimensions match designs within 0.5mm

Exercise A2: The Customizable Keycap Set (Intermediate)

Objective: Design a family of keycaps for a custom keyboard with text, icons, and different profiles.

Requirements:

- 5+ keycap sizes (mm 12-28)
- Support embossed text and numbers
- Optional: icon cutouts
- All printable without supports

Challenge: Use parametric code to generate all 5 caps from a single design[225]:

```
// Parametric Keycap Generator v1
// PARAMETERS TO CUSTOMIZE
cap_size = 18; // mm - change for different key sizes
cap_height = 12; // mm
wall = 1.2; // mm
text_char = "A"; // Character to emboss
emboss_depth = 0.8; // mm
// Derived calculations
inner_size = cap_size - 2*wall;
module keycap() {
  // Hollow shell
  difference() {
    cube([cap_size, cap_size, cap_height], center=false);
    translate([wall, wall, wall])
      cube([inner_size, inner_size, cap_height], center=false);
  }
}
module emboss() {
  translate([cap_size/2, cap_size/2, cap_height - 0.01])
    linear_extrude(height=emboss_depth)
      text(text_char, size=cap_size*0.5,
           halign="center", valign="center", $fn=32);
}
// Main assembly
union() {
  keycap();
  emboss();
}
```

Tasks:

[225] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

1. Create variants with `cap_size` = 12, 16, 18, 20, 24 mm
2. Build each variant and verify emboss quality
3. Test printability: print 2-3 variants
4. Measure final dimensions and compare to design

Documentation Required:

- Parameter table showing all 5 variants
- Print time estimates for each
- Post-processing notes (support removal, surface finishing)

Exercise A3: The Stackable Storage System (Advanced)

Objective: Design a modular storage system where bins stack securely and dividers are customizable.

Requirements:

- Bins must stack without binding
- Dividers must be optional and repositionable
- 3+ configurations (small/medium/large)
- Tolerance management documented[226]

Key Features:

```
// Advanced Stackable Bin System
// PARAMETERS
bin_width = 80;
bin_depth = 120;
bin_height = 60;
wall_thick = 2;
rim_height = 3;
stack_clearance = 0.6; // KEY TOLERANCE PARAMETER
divider_thickness = 1.5;
num_dividers = 2; // 0 for no dividers
// Tolerance-critical calculation
rim_inner_width = bin_width - 2*(wall_thick + stack_clearance);
rim_inner_depth = bin_depth - 2*(wall_thick + stack_clearance);
module bin_body() {
  difference() {
    cube([bin_width, bin_depth, bin_height]);
    translate([wall_thick, wall_thick, wall_thick])
      cube([bin_width - 2*wall_thick,
           bin_depth - 2*wall_thick,
           bin_height - wall_thick]);
  }
}
module stacking_rim() {
  difference() {
    translate([0, 0, bin_height])
      cube([bin_width, bin_depth, rim_height]);
    translate([wall_thick + stack_clearance,
              wall_thick + stack_clearance,
              bin_height])
      cube([rim_inner_width, rim_inner_depth, rim_height]);
  }
}
module dividers() {
  if (num_dividers > 0) {
```

[226]OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

```
    for (i = [1 : num_dividers]) {
      y = wall_thick + (i * (bin_depth - 2*wall_thick) / (num_dividers + 1));
      translate([wall_thick, y, wall_thick])
        cube([bin_width - 2*wall_thick, divider_thickness, bin_height - 2* →
             ↪ wall_thick]);
    }
  }
}
// Main assembly
union() {
  bin_body();
  stacking_rim();
  dividers();
}
```

Tasks:

1. Print 2 identical bins and test stacking
2. Create 3 configurations: Small (60x80x40), Medium (80x120x60), Large (120x160x80)
3. Test with different `stack_clearance` values: 0.4, 0.6, 0.8mm
4. Document which clearance works best in practice[227]
5. Print a large bin with 3 dividers and test storage capacity

Documentation Required:

- Tolerance testing matrix with measurements
- Assembly instructions with photos/descriptions
- Recommendations for printer calibration

Advanced Geometry Techniques: Hull and Minkowski Operations

Beyond basic primitives and boolean operations, professional designs often use hull and Minkowski operations to create smooth, rounded, and organic shapes. These are essential for creating high-quality, manufacturable designs.

Example 1: Hull Operations for Organic Shapes

The `hull()` function creates the convex hull of shapes-imagine wrapping them in plastic wrap. This is perfect for creating smooth, organic forms:

```
// Birdhouse using hull() - from Simplifying 3D Printing textbook
// Demonstrates combining simple shapes into organic forms
$fn = 100;
module generate_bird_shape() {
    hull() {
        // Create organic shape by blending multiple spheres
        translate([0, 0, 90])
        resize([180, 180, 180])
        minkowski() {
            cube([10, 10, 10], center = true);
            rotate([0, 0, 45])
            cube([25, 25, 55], center = true);
            cylinder(h = 40, d1 = 30, d2 = 2, center = true);
```

[227] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

```
        }
        // Second shape - top of birdhouse
        translate([0, 0, 150])
        sphere(65);
    }
}
// Hollow out the interior
module birdhouse_hollow() {
    difference() {
        generate_bird_shape();
        // Entrance hole
        translate([0, 100, 120])
        rotate([90, 0, 0])
        cylinder(h = 200, d = 80, center = true, $fn = 64);
        // Hollow interior
        translate([0, 0, -2])
        scale([0.95, 0.95, 1])
        generate_bird_shape();
    }
}
birdhouse_hollow();
```

Key insights:

- `hull()` automatically creates a smooth envelope around shapes
- Blending multiple spheres creates organic, curved surfaces
- Combining `hull()` with `minkowski()` creates sophisticated forms
- Great for product design, containers, and artistic shapes

Example 2: Minkowski Operations for Rounded Edges

The `minkowski()` function adds a shape to every point of another shape. This is perfect for creating rounded edges and fillets without complex calculations:

```
// Parametric Phone Stand with Minkowski Rounding
// Demonstrates edge rounding for professional appearance
$fn = 100;
thickness = 4;
width = 70;
depth = 90;
angle = 65;
lipheight = 12;
filletr = 6; // Fillet radius
module plate(w, d, t) {
    cube([w, d, t], center = false);
}
module rounded_plate(w, d, t, r) {
    // Use minkowski to add rounded edges
    minkowski() {
        cube([w - 2*r, d - 2*r, t], center = false);
        cylinder(h = 0.01, r = r, $fn = 40);
    }
}
// Base plate with rounded edges
module base() {
    rounded_plate(width, depth, thickness, filletr);
}
// Back plate at angle with rounded edges
module back() {
    rotate([angle, 0, 0])
    rounded_plate(width, depth, thickness, filletr);
}
// Lip to hold device
module lip() {
    translate([0, depth - 8, thickness])
    cube([width, 8, lipheight], center = false);
}
// Main assembly
union() {
    base();
    back();
    lip();
}
```

Comparison:

Method	Com-plexity	Speed	Quality	Use Case
Sharp edges	Simple	Fast	Angular, technical	Mechanical parts
`hull()` for rounding	Medium	Medium	Smooth, organic	Consumer products
`minkowski()` for fillets	Medium	Medium	Professional	Finished goods
Manual fillet modules	Complex	Slow	Precise	Custom requirements

Example 3: Combining Hull and Minkowski

Create sophisticated shapes by combining multiple techniques:

```
// Advanced technique: Hull + Minkowski for professional design
$fn = 100;
module advanced_connector() {
    minkowski() {
        // Base shape with hull
        hull() {
            cube([10, 10, 30], center = true);
            translate([20, 0, 0])
            cylinder(h = 30, d = 8, center = true);
        }
        // Add small sphere for fillet
        sphere(r = 2);
    }
}
// Use in assembly
union() {
    advanced_connector();
    translate([50, 0, 0])
    advanced_connector();
}
```

When to use each technique:

- Use `hull()` when you need organic, flowing shapes
- Use `minkowski()` when you need consistent rounding around edges
- Combine them for maximum design flexibility
- Test both approaches and choose based on render time and output quality

Design Best Practices

1. Start simple: Create basic geometry first, then add rounding
2. Test early: Render at low `$fn` during design, high `$fn` before export
3. Document parameters: Make rounding radius and other values adjustable
4. Validate manifold: Always run `3dm describe` after using hull/minkowski
5. Consider performance: These operations can slow down preview significantly

Exercise Set B: Common Problems and Solutions

B1: Non-Manifold Geometry

Problem: Model renders but slicer shows errors or generates invalid G-code

Diagnosis:

```
3dm describe your_model.scad
# Look for messages like "non-manifold edges" or "thickness near zero"
```

Common Causes and Fixes:

Issue	Cause	Fix
Coincident faces	Shapes touching exactly	Add 0.001 offset: `translate([0, 0, 0.001])`
Zero-thickness walls	Wall too thin to render	Increase wall: `wall = 2.0` instead of `wall = 0.5`
Incomplete shape	Missing face in difference	Check boolean operations have complete shells
Self-intersecting faces	Shape overlaps itself	Simplify geometry, use `hull()` instead of union

Fix Example:

```
// BEFORE (often fails)
difference(){
  cube([20, 20, 20]);
  sphere(r=10);
}
// AFTER (usually works)
difference(){
  cube([20, 20, 20]);
  translate([10, 10, 10.001]) // Offset to prevent coincidence
    sphere(r=10, $fn=32);
}
```

B2: Print Failures

Problem: Print starts but fails partway through

Diagnosis:

```
# Check G-code for errors
3dm slice your_model.scad

# Verify model in slicer before printing
# Preview layer-by-layer
```

Common Causes:

Symptom	Cause	Prevention
Nozzle clogs after 10 min	Too fast extrusion speed	Reduce speed in slicer
Parts separate from bed	Poor first layer adhesion	Check bed level, clean bed
Melted plastic strands	Retraction not working	Verify retraction in slicer
Model warping	Thermal stress, no cooling	Cool model, improve ventilation
Supports fail to remove	Too thin or fused to model	Thicken supports, adjust angle

Validation Checklist Before Printing:

```
#!/bin/bash
# pre_print_check.sh - Validate before printing

echo "=== Pre-Print Validation ==="

# 1. Check for non-manifold geometry
echo "Checking geometry..."
3dm describe src/main.scad | grep -i "non-manifold"

# 2. Check wall thickness
echo "Checking wall thickness > 1mm..."
# (This requires slicer analysis - do manually)

# 3. Generate preview
echo "Generating 2D preview..."
3dm preview src/main.scad

# 4. Estimate print time
3dm slice src/main.scad
echo "Generated G-code: build/main.gcode"

echo "=== Ready to print? ==="
```

B3: Dimensional Inaccuracy

Problem: Printed part measures 0.5-2mm different from design

Cause: Printer calibration, shrinkage, or design tolerance issues[228]

Diagnosis:

```
# After printing, measure 3 locations and average
# Compare to design parameters
```

[228] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

Solution Process:

```
Step 1: Measure printed part (3 locations)
       Design: 50mm
       Measured: 49.7mm, 49.8mm, 49.6mm
       Average: 49.7mm (0.3mm small)

Step 2: Calculate correction factor
       Correction = 50.0 / 49.7 = 1.0060 (0.6% larger)

Step 3: Apply to design
       New param = old_param * 1.0060

Step 4: Reprint and verify
```

Prevention:

- Use standard calibration models (benchy, calibration cube)
- Document printer baseline dimensions
- Apply material-specific shrinkage factors

Exercise Set C: Validation and Documentation

C1: Design Review Checklist

After completing any design, validate using this checklist:

```
# Design Review Checklist

## Geometry Validation
- [ ] Model builds without errors (`3dm build`)
- [ ] No non-manifold edges (`3dm describe` shows no warnings)
- [ ] All walls 1.5mm thick
- [ ] No floating parts or disconnected geometry
- [ ] All features clearly serve a purpose

## Parametric Design
- [ ] All dimensions are parameters (no hard-coded values)
- [ ] Parameter names are descriptive (e.g., `wall_thickness` not `w`)
- [ ] Parameters have units comments (e.g., `wall = 2; // mm`)
- [ ] Derived values calculated from parameters (e.g., `inner = outer - 2* wall`)

## Printability
- [ ] Overhangs 45 or have support strategy
- [ ] No fine details < 0.5mm
- [ ] Assembly features tested (snap-fits, interlocks)
- [ ] Post-processing needs documented

## Documentation
- [ ] README describes design purpose and key parameters
- [ ] Usage examples provided
- [ ] Known limitations documented
- [ ] Assembly instructions included (if multi-part)

## Accessibility
- [ ] Model dimensions available in text form
- [ ] AI description (`3dm describe`) is useful
- [ ] Assembly process described non-visually
- [ ] No features rely solely on visual inspection
```

C2: Troubleshooting Documentation Template

After solving any issue, document it:

```
# Troubleshooting Record

## Issue: [Describe the problem]
Date: YYYY-MM-DD
Project: [Project name]

## Symptoms
- [What did you observe?]
- [When did it happen?]

## Root Cause
[Analysis of why it happened]

## Solution Applied
[Exact steps to fix]

## Verification
[How did you confirm it worked?]

## Prevention
[How to avoid this in future designs]

## References
- [Related documentation]
- [Similar issues]
```

Checkpoint

- After Exercise A1, you have a working phone stand
- After Exercise A2, you have a parametric keycap set
- After Exercise A3, you have a stackable bin system with tolerance data
- After Exercise B, you understand common failure modes
- After Exercise C, you have documentation habits

Quiz - Lesson 3dMake.10 (10 questions)

1. What is the first diagnostic step when a model fails to slice[229]?
2. Explain the 0.001 offset rule and when to apply it[230].
3. How would you verify that a wall is thick enough before printing[231]?
4. What measurement method would you use to validate dimensional accuracy[232]?
5. Describe what happens when stack_clearance is too small or too large[233].
6. True or False: Parametric design makes it harder to troubleshoot issues.
7. How would you document a design so that others can modify and reprint it[234]?

[229] 3D Printing Troubleshooting Guide - https://www.prusa3d.com/support/
[230] 3D Printing Troubleshooting Guide - https://www.prusa3d.com/support/
[231] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[232] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[233] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[234] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

8. What should you check in a slicer preview before sending a file to print[235]?
9. How would you use `3dm describe` to find design flaws[236]?
10. Describe a complete workflow from problem diagnosis to solution verification[237].

Extension Problems (10)

1. Complete all three exercises (A1, A2, A3) and document results[238].
2. Print and test 2+ variants of your phone stand; measure and compare[239].
3. Conduct a tolerance sensitivity study: test bins with 3+ clearance values[240].
4. Create a troubleshooting guide for your specific printer: common issues and fixes[241].
5. Design a validation workflow: automated checks before printing[242].
6. Build a design documentation database: parameter ranges, material recommendations, print settings[243].
7. Create before/after case studies of design problems and solutions[244].
8. Develop a printer calibration procedure and document baseline dimensions[245].
9. Design a quality assurance system: define metrics, measurement methods, acceptance criteria[246].
10. Write a comprehensive troubleshooting and maintenance guide: diagnostics, solutions, prevention strategies, and accessibility considerations.

References

Diagnostic Checklist for 3D Printing

Use this comprehensive checklist to systematically diagnose and troubleshoot printing issues.

Quick Diagnosis Flowchart

```
Print Problem?
|
```

[235] 3D Printing Troubleshooting Guide - https://www.prusa3d.com/support/
[236] 3DMake Documentation - https://github.com/tdeck/3dmake/wiki/Troubleshooting
[237] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[238] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[239] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[240] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[241] 3D Printing Troubleshooting Guide - https://www.prusa3d.com/support/
[242] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[243] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[244] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[245] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
[246] OpenSCAD Best Practices - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ

```
+---- [Before Print] Issues?
| +---- Filament won't load -> CHECK: Temperature, drive gear, nozzle
| +---- Printer won't heat -> CHECK: Power, temperature sensor, firmware
| +---- Bed not level -> CHECK: Leveling routine, bed surface
|
+---- [First Layer] Issues?
| +---- Won't stick -> CHECK: Bed temperature, cleanliness, nozzle height
| +---- Too squished -> CHECK: Nozzle height, bed temperature
| +---- Gaps/uneven -> CHECK: Bed levelness, hot end alignment
|
+---- [Mid-Print] Issues?
| +---- Stops extruding -> CHECK: Clog, temperature drop, jam
| +---- Layers shift -> CHECK: Loose belts, mechanical bind
| +---- Print wobbles -> CHECK: Build plate, print stability
|
+---- [Print Quality] Issues?
  +---- Weak/brittle -> CHECK: Temperature, flow rate, layer adhesion
  +---- Rough surface -> CHECK: Flow rate, speed, temperature
  +---- Warped -> CHECK: Bed temperature, cooling rate, material
```

Pre-Print Diagnostics

Category A: Power & Connectivity Checklist:

- ☐ Printer powered on
- ☐ LED indicators showing normal status
- ☐ USB cable connected (if applicable)
- ☐ No error codes displaying
- ☐ Display/interface responding to input

If failed:

1. Check power outlet and cable
2. Verify power supply specifications (voltage, current)
3. Test with different power outlet
4. Try power-cycling (off 30 sec, on)
5. Check for blown fuses inside printer

Category B: Temperature System Heating Element Status:

- ☐ Hot end temperature rises when heating commanded
- ☐ Bed temperature rises when heating commanded
- ☐ Temperature readings stable (not fluctuating 5C+)
- ☐ No error messages during heating

Measurement Method:

```
1. Set hot end to 200C, observe rise
  - Expected time to reach: 2-4 minutes
  - Steady rise without plateau: Good
  - Plateau before reaching: Problem (see below)
2. Set bed to 60C, observe rise
  - Expected time to reach: 5-10 minutes
  - Stable at target: Good
```

If heating slow/incomplete:

- ☐ Verify target temperature was set
- ☐ Check heating element firmware settings
- ☐ Test thermal sensor connectivity
- ☐ Inspect heating element for damage
- ☐ Measure electrical resistance of heaters

Temperature Stability Test:

```
1. Heat to target temperature
2. Wait 10 minutes for stabilization
3. Record temperature every minute
4. Calculate range (max - min)
Results:
- +/-2C range: Excellent
- +/-5C range: Acceptable
- +/-10C range: Marginal
- >+/-10C range: Problem
```

Category C: Mechanical Systems Movement Diagnostics:

Manual Axis Movement:

```
1. Disable motors (if possible)
2. Manually move each axis
3. Record observations:
   - X-axis: (smooth/rough/stuck)
   - Y-axis: (smooth/rough/stuck)
   - Z-axis: (smooth/rough/stuck)
```

Expected results: Smooth, no grinding sounds

If rough/stuck:

- ☐ Check for visible obstructions
- ☐ Inspect rails for debris
- ☐ Verify pulleys turn freely
- ☐ Check belt tension (if accessible)
- ☐ Lubricate dry joints

Powered Movement Test:

```
1. Position nozzle at center
2. Command X+10mm movement
3. Verify nozzle moved ~10mm
4. Repeat for Y+10mm and Z+5mm
5. Check for skipping or missed steps
```

Category D: Leveling & Alignment Build Plate Leveling Test:

Paper Method (Most Common):

```
1. Heat bed to printing temperature
2. Heat nozzle to printing temperature
3. Position nozzle at first corner
4. Adjust leveling screw until paper drags slightly
5. Move to next corner and repeat
6. Repeat for all 4-9 corners
7. Do center point check last
```

```
Target: Consistent slight paper drag all points
```

Leveling Validation:

- ☐ Level at 4 corners
- ☐ Level at bed center
- ☐ No high/low points
- ☐ Nozzle doesn't hit bed at any point
- ☐ Consistent first-layer appearance across bed

Category E: Filament & Extruder Filament Quality Check:

- ☐ Filament diameter consistent (visually inspect ~50cm)
- ☐ No visible cracks or damage
- ☐ Spool rotates freely without binding
- ☐ Filament path clear to extruder
- ☐ No tangles in filament path

Extruder Test:

```
1. Heat to printing temperature
2. Remove print head (if removable)
3. Push 10-20mm of filament through manually
4. Feel resistance during push
5. Observe material exits cleanly
Expected: Smooth push, consistent extrusion
```

Filament Loading Test

Test Sequence:

```
1. Heat extruder to material temp
2. Load filament into extruder
3. Watch for material at nozzle tip
Timeline:
- 0-10 sec: Filament engaging
- 10-30 sec: Moving through hot end
- 30-60 sec: Should appear at nozzle tip
- >60 sec: Possible partial clog
If fails:
-> See "Filament Won't Load" troubleshooting
```

First Layer Diagnostics

Layer Appearance Test After printing first 5-10 layer heights, evaluate:

Appearance	Issue	Action
Wavy/embossed	Bed not level or too close	Relevel bed
Gaps between lines	Nozzle too high	Lower Z-offset
Completely squished	Nozzle too low	Raise Z-offset
Partial adhesion	Bed too cool or dirty	Clean bed, increase temp

Appearance	Issue	Action
Consistent squish/lines	Correct	Continue print

Mid-Print Issue Diagnostics

Extrusion Failure Checklist When extrusion stops during print:

Immediate Actions:

- ☐ Pause print (don't stop)
- ☐ Listen for extruder sounds (grinding = jam)
- ☐ Feel nozzle carefully (if cooled slightly)
- ☐ Observe filament in extruder (is it feeding?)

Diagnostic Decision:

```
Is filament stuck in extruder?
+---- YES -> Nozzle clog likely
| -> See Nozzle Clog section (common_issues_and_solutions.md)
| -> Try: Cold pull, retract, clean
|
+---- NO -> Filament loading issue
        -> Is spool binding? -> Fix spool rotation
        -> Is path blocked? -> Clear obstruction
        -> Is drive gear slipping? -> Clean/tension drive gear
```

Mechanical Issue Diagnostics When movement sounds wrong:

Listen for:

- Grinding/grating: Bearing issue or obstruction
- Clicking/skipping: Lost steps or over-torque
- Squealing: Lubrication needed
- Silence (but no movement): Stalled motor

Diagnosis Method:

```
1. Pause print
2. Manually move suspected axis
3. Record resistance type:
   - Smooth: Normal
   - Rough: Bearing/alignment problem
   - Stuck: Mechanical bind
4. Visually inspect that axis
```

Layer Quality Diagnostics

Visual Inspection During Print Every 30 minutes of printing, check:

```
[ ] Layer alignment (no X/Y shifting)
[ ] Material flow (consistent lines, not thin or thick)
[ ] Surface appearance (smooth, not rough)
[ ] Support structure (if used, printing properly)
[ ] No material strings between features
[ ] Consistent layer heights visible
```

Dimensional Accuracy Diagnostics

After print completes and cools (24 hours):

Precision Measurement Materials Needed:

- Digital calipers (+/-0.05mm accuracy)
- Ruler (for larger dimensions)
- Notepad for recording

Measurement Protocol:

```
1. Measure each dimension 3 times at different locations
2. Calculate average
3. Compare to design dimension
4. Calculate deviation percentage
Formula:
Deviation % = ((Measured - Design) / Design) x 100%
Example:
- Design: 20.0mm
- Measured: 19.8mm
- Deviation: ((19.8 - 20.0) / 20.0) x 100% = -1%
```

Tolerance Evaluation

Tolerance	Pass/Fail	Action
+/-0.5mm or better	PASS	No adjustment needed
+/-0.5-1mm	MARGINAL	Document and monitor
>+/-1mm	FAIL	Adjust flow/calibration

Environmental Diagnostics

When quality varies between prints:

Check Conditions:

- ☐ Room temperature stable (+/-5C?)
- ☐ Humidity reasonable (30-60%?)
- ☐ No drafts from windows/AC near printer
- ☐ Consistent vibration level (no external impact)
- ☐ Same filament spool/batch used
- ☐ Same slicer settings applied

Environmental Log:

```
Date: Time: Temp: C Humidity: %
Print Duration: Result Quality: Poor/Fair/Good/Excellent
Notes:
```

Troubleshooting Decision Tree

Start here for systematic diagnosis:

```
+---- Printer won't start?
| +---- Check: Power, connections, firmware
|
+---- Heating won't work?
| +---- Check: Temperature sensor, firmware, heater element
|
+---- Won't home/move?
| +---- Check: Endstops, mechanical bind, motors, belts
|
+---- First layer fails?
| +---- Check: Bed level, nozzle height, cleanliness, temperature
|
+---- Filament won't feed?
| +---- Check: Temperature, nozzle, drive gear, clog
|
+---- Extrusion stops mid-print?
| +---- Extruder grinding? -> Clog (cold pull or replace nozzle)
| +---- Filament slack? -> Drive gear or load issue
| +---- No sounds? -> Temperature drop or firmware issue
|
+---- Print quality poor?
| +---- Weak/thin? -> Increase flow/temp/slow down
| +---- Rough/bloated? -> Decrease flow/temp/speed up
| +---- Warped? -> Lower bed temp, faster cooling
| +---- Strings? -> Increase retraction, lower temp
|
+---- Dimensions wrong?
   +---- Check: Flow rate calibration, printer accuracy limits
```

Diagnostic Report Template

Use when seeking help:

```
DIAGNOSTIC REPORT
================
Printer Model:
Problem Description:
When it occurs: (always/sometimes/first 5 layers, etc)
Recent changes:
DIAGNOSTICS PERFORMED:
[ ] Power/connectivity verified
[ ] Temperatures verified
[ ] Mechanical movement tested
[ ] Bed leveling checked
[ ] Filament loading tested
[ ] First layer inspected
[ ] Print quality evaluated
Key Findings:
1.
2.
3.
Attempted Solutions:
1.
2.
Result: (Solved/Partial/Ongoing)
```

Last Diagnostic Date: Issue Resolved:
Diagnostic Performed By:

Measurement Calibration Guide

Ensure accurate dimensions in your 3D printed parts through proper calibration and verification procedures.

Why Calibration Matters

Printed dimensions often deviate from designed dimensions due to:

- Nozzle width variations
- Extrusion flow rate inconsistencies
- Plastic shrinkage during cooling
- Slicer interpretation differences
- Material-specific shrinkage (ABS can shrink 0.3-1%)

Typical tolerances without calibration: +/-0.3-0.5mm Typical tolerances with calibration: +/-0.1-0.2mm

Pre-Print Calibration

1. Nozzle Diameter Verification What to verify: Is your nozzle actually 0.4mm?

Test:

1. Create a simple box in OpenSCAD: 10mm x 10mm x 5mm, solid
2. Slice with default 0.4mm line width
3. Print using 100% flow rate
4. Measure printed box

Adjustment:

- If too small: Nozzle might be partially clogged (clean)
- If significantly different from 0.4mm: Replace nozzle

2. E-Steps Calibration (Extrusion Rate) What to verify: Does the extruder push the correct amount of filament?

Method:

1. Heat extruder to printing temperature
2. Mark filament 100mm from extruder entrance with marker
3. Command extrusion of 100mm in firmware (G-code: `G1 E100 F100`)
4. Measure actual distance filament moved

Formula:

```
New E-steps = Current E-steps x (100mm / Actual distance moved)
```

Example:

- Current setting: 93 steps/mm
- Commanded: 100mm
- Actual: 92mm moved
- New: 93 x (100/92) = 101 steps/mm

How to apply (varies by printer):

- Marlin: `M92 E101` then `M500` to save
- Klipper: Update configuration and restart

3. First Layer Height Verification What to verify: Is first layer height optimal?

Test: Print a simple single-layer square (just base layer)

Measurements:

- Too high (>0.3mm): Poor layer adhesion
- Too low (<0.1mm): Nozzle scratches bed, plastic squeezed
- Optimal: 0.2-0.25mm (roughly paper thickness)

Adjustment: Use bed leveling or Z-offset:

- If too high: Reduce Z-offset by 0.05mm
- If too low: Increase Z-offset by 0.05mm
- Test between adjustments

Dimension Calibration Process

Standard XY Calibration Test Goal: Create parts with precisely measured dimensions

Test File (OpenSCAD):

```
// Create calibration cube
cube_size = 20; // 20mm cube
wall_thickness = 2;

difference() {
    cube([cube_size, cube_size, cube_size], center=true);
    cube([cube_size - 2*wall_thickness,
          cube_size - 2*wall_thickness,
          cube_size + 1], center=true); // Top open
}
```

Print Instructions:

- Use standard settings (your normal layer height, speed, temp)
- Print with 100% flow rate
- Allow complete cooling (2+ hours)

Measurement Procedure:

1. Measure internal dimensions (hollow part) in 3 locations each axis
2. Calculate average internal width: `avg_internal`
3. Expected internal: `20 - 2xwall_thickness = 16mm`

Calibration Formula:

```
Flow rate adjustment = Expected internal / Actual internal x 100%

Example:
- Expected: 16.00mm
- Actual: 15.75mm
- Adjustment: (16.00 / 15.75) x 100% = 101.6%
- Set flow to: 101.6% in slicer
```

Z-Height Calibration Goal: Verify layer heights are accurate

Test: Print calibration tower with varying layer heights

Tower Specifications:

- 20mm x 20mm square base
- Height: 40mm
- Layers: Print at 0.2mm nominal

Measurements:

- Stack digital calipers on layers and measure height
- Calculate average layer thickness
- Compare with intended 0.2mm

Adjustment: If actual layer height differs:

```
New Z-scale = Intended height / Actual height

Example:
- Intended: 0.2mm per layer
- Actual: 0.195mm per layer
- Adjustment: 0.2 / 0.195 = 1.026 (increase Z by 2.6%)
```

Tolerance Measurement Matrix

Critical Measurements to Track

Measurement	Method	Tolerance	Frequency
Wall thickness	Calipers (multiple spots)	+/-0.1mm	Every print
Hole diameter	Calipers or gauge	+/-0.1-0.2mm	Every print
Overall dimensions	Ruler/calipers	+/-0.2mm	Monthly
Layer height	Stack on calipers	+/-0.02mm	Monthly
Vertical dimensions	Measure sides	+/-0.1mm	Every print

Advanced Calibration

Shrinkage Compensation Different materials shrink differently after cooling:

Material	Typical Shrinkage	Compensation
PLA	0.3-0.5%	Usually acceptable, no action
PETG	0.5-1%	Scale design up by 0.5-1% if critical
ABS	0.8-1.5%	Scale design up by 1% minimum
TPU	1-2%	Significant - scale up 1-2% for critical dimensions

How to apply in design:

```
// In OpenSCAD, scale critical dimensions
final_size = 20;
material_shrinkage = 1.01; // 1% shrinkage
designed_size = final_size * material_shrinkage;
```

Bed Temperature Compensation Different bed temperatures affect final dimensions:

ABS on cold bed (50C) vs warm bed (100C):

- Cold bed: Faster cooling, less shrinkage (but poor adhesion)
- Warm bed: Slower cooling, more shrinkage (better adhesion)
- Difference: Can be 0.2-0.3% in dimensions

Solution: Standardize bed temperature for repeatable results

Environmental Factors

Factor	Effect	Mitigation
Room temperature	Affects cooling rate	Maintain 20-22C
Humidity	Affects material properties	Keep 40-60% RH
Air flow	Inconsistent cooling	Avoid drafts near printer
Time of day	Material temperature varies	Print at consistent times

Quick Calibration Checklist

Before First Print with New Settings

- ☐ E-steps calibration complete
- ☐ First layer height verified
- ☐ Nozzle diameter confirmed
- ☐ Test print completed and measured

Monthly Maintenance

- ☐ Calibration cube printed and measured
- ☐ Flow rate adjusted if needed
- ☐ Layer height verified
- ☐ Temperature consistency checked

When Dimensions Are Critical

- ☐ Printed test part, let cool 24+ hours
- ☐ Measured in multiple locations
- ☐ Calculated average deviation
- ☐ Flow rate adjusted accordingly
- ☐ Re-printed and verified

After Any Changes

- ☐ Nozzle replacement -> Re-verify nozzle diameter
- ☐ Bed leveling -> Re-verify first layer
- ☐ Temperature changes -> Test print required
- ☐ Material change -> Full calibration recommended

Measurement Tools Needed

Tool	Cost	Accuracy	Use
Digital Calipers	$5-15	+/-0.05mm	Primary measurements
Steel Ruler	$3-10	+/-1mm	Quick rough checks
Vernier Calipers	$10-30	+/-0.05mm	Precision work
Micrometer	$20-50	+/-0.01mm	Critical tolerances
Layer Height Gauge	DIY or $5-10	+/-0.05mm	Layer verification

Recommendation: Start with digital calipers (most versatile and affordable)

Troubleshooting Calibration Issues

Problem: Measurements still inconsistent after calibration

- Check if bed is level (temperature affects levelness)
- Verify material is dry (moisture affects dimensions)
- Ensure ambient temperature is stable
- Try printing on different bed locations

Problem: Can't achieve target dimensions

- Nozzle may be damaged/worn (try replacement)
- Printer may have fundamental hardware issues

- Review mechanical components (belts, screws)
- Consider printer calibration limits

Problem: Dimensions drift over time

- Printer thermal properties changing
- Nozzle wearing out (gradually gets smaller)
- Bed surface degrading
- Normal wear - recalibrate quarterly

Reference: Standard Test Models

These models are helpful for calibration:

1. Calibration Cube (20mm hollow) - Overall accuracy
2. Tolerance Test Box (various hole sizes) - Hole accuracy
3. Layer Tower (graduated heights) - Layer consistency
4. Thin Wall Test (walls 1-5mm) - Wall thickness accuracy

Last Calibration Date: _
Printer Model: _
Current E-Steps: _
Current Flow Rate: _
Materials Calibrated For: _

Measurement Worksheet

Student Name: _ Date: _ Project / Assignment: _

How to Use This Worksheet

1. Measure each feature three times and record all three values
2. Calculate the average: add the three values and divide by 3
3. Round to one decimal place (e.g., 23.4 mm)
4. Use the average in your OpenSCAD code - not a single measurement

Units: All measurements in millimeters (mm) unless otherwise noted.

Object 1

Object description: _

Feature	What You're Measuring	M1 (mm)	M2 (mm)	M3 (mm)	Average (mm)
Length (X)					
Width (Y)					
Height (Z)					

Feature	What You're Measuring	M1 (mm)	M2 (mm)	M3 (mm)	Average (mm)
Feature 4					
Feature 5					

Notes / sketches:

```
(describe the object here - label which direction is X, Y, Z)
```

Object 2

Object description: _

Feature	What You're Measuring	M1 (mm)	M2 (mm)	M3 (mm)	Average (mm)
Length (X)					
Width (Y)					
Height (Z)					
Feature 4					
Feature 5					

Notes / sketches:

```
(describe the object here)
```

Object 3

Object description: _

Feature	What You're Measuring	M1 (mm)	M2 (mm)	M3 (mm)	Average (mm)
Length (X)					
Width (Y)					
Height (Z)					
Feature 4					
Feature 5					

Notes / sketches:

```
(describe the object here)
```

Object 4

Object description: _

Feature	What You're Measuring	M1 (mm)	M2 (mm)	M3 (mm)	Average (mm)
Length (X)					
Width (Y)					
Height (Z)					
Feature 4					
Feature 5					

Notes / sketches:

```
(describe the object here)
```

Object 5

Object description: _

Feature	What You're Measuring	M1 (mm)	M2 (mm)	M3 (mm)	Average (mm)
Length (X)					
Width (Y)					
Height (Z)					
Feature 4					
Feature 5					

Notes / sketches:

```
(describe the object here)
```

Accuracy Check

After completing all measurements, compare your averages with a partner who measured the same objects.

Object	My Average (mm)	Partner's Average (mm)	Difference (mm)	Within 1 mm?
1				
2				

Object	My Average (mm)	Partner's Average (mm)	Difference (mm)	Within 1 mm?
3				
4				
5				

If any difference is greater than 1 mm, remeasure together and find the source of the discrepancy.

Percent Error (Optional / Extension)

If your instructor provides the ”true” dimension of an object (measured with a reference instrument), you can calculate your percent error:

Formula: `% error = (|your average  true value| / true value) x 100`

Object	Your Average	True Value	Difference	% Error

A percent error under 2% is excellent for caliper work at this level.

Reflection

Answer in complete sentences.

1. Which measurement was most difficult to take and why?
2. Did your three measurements for any feature vary significantly? What might cause variation between repeated measurements?
3. If you were designing an object in OpenSCAD that needed to fit over one of these objects, which measurement would you use - your smallest, your largest, or your average? Why?

Common 3D Printing Issues and Solutions

Comprehensive troubleshooting guide for diagnosing and fixing common 3D printing problems.

Pre-Print Issues

Problem: Filament Won't Load Symptoms:

- Extruder clicks/grinds but no filament moves

- Nozzle temperature displays but stays clean
- No material extrudes when manual extrude command sent

Diagnosis Checklist:

☐ Nozzle temperature high enough? (Check material specs)
☐ Filament path clear of obstructions?
☐ Drive gear has grip on filament (not polished smooth)?
☐ Extruder tension set correctly?

Solutions:

1. Increase nozzle temperature (too cold = harder extrusion)
2. Clean extruder drive gear - brush with wire brush to restore grip
3. Check filament diameter - should be 1.75mm (+/-0.03mm)
4. Verify extruder tension - should grip firmly but not crush filament
5. Inspect nozzle - may be partially clogged (see nozzle clog section)

Problem: Warped/Damaged Filament Symptoms:

- Filament appears bent or kinked on spool
- Filament diameter inconsistent
- Melted spots on filament surface

Diagnosis Checklist:

☐ Was filament stored properly (dry, cool)?
☐ Spool has been sitting unused?
☐ Transport/handling damage visible?

Solutions:

1. Discard damaged section - cut off ~30cm if kinked
2. Increase humidity - try drying filament in low oven (50C for 2 hours PLA)
3. Check storage - PLA especially absorbs moisture
4. Replace spool - if damage is extensive

Print Quality Issues

Problem: Poor Bed Adhesion (First Layer Lifts) Symptoms:

- Print lifts off bed during first layer
- Material rolls up at edges
- Print comes free during printing

Diagnosis Checklist:

☐ Build plate level? (Should drag paper at all points)
☐ Nozzle at correct height for first layer?
☐ Bed clean and free of dust/oil?
☐ Bed temperature adequate?

Solutions:

1. Re-level build plate - cold plate adjustment for accuracy
2. Clean build plate - wipe with isopropyl alcohol
3. Lower nozzle (if too high) - reduce Z by 0.05-0.1mm
4. Increase bed temperature - add 5-10C
5. Use adhesion aids:
    - PLA: Painter's tape, glue stick, or bare clean plastic
    - PETG: Textured bed or thin glue layer
    - ABS: Heated bed + blue tape + ABS slurry (acetone + scraps)
    - TPU: Clean bed, possibly elevated temperature

Problem: Nozzle Clog Symptoms:

- No extrusion after initial layers
- Consistent gap between nozzle and print
- Pressure building in nozzle (smoking or popping)
- Clicking sounds from extruder

Diagnosis Checklist:

☐ Filament jam visible in extruder?
☐ Nozzle temperature high enough?
☐ Print moved away from nozzle?
☐ Recent failed print or debris?

Solutions (Increasing Intensity):

1. Immediate - Cold Pull:
    - Heat nozzle to 200C
    - Grab filament and give firm pull while cooling
    - Repeat 5-10 times
    - May extract filament with debris
2. Hot Swap:
    - Heat nozzle to printing temperature
    - Remove extruder completely
    - Use small drill bit (0.3-0.4mm) to carefully poke from top
    - Don't force - may damage nozzle
3. Soak & Poke:
    - Remove nozzle with wrench while hot
    - Soak in heated acetone or strong cleaner (20-30 min)
    - Use ultrasonic cleaner if available
    - Use thin needle to clear passage
4. Replacement:
    - If clog persists, nozzle may be damaged internally

  - Replace with new nozzle (usually $3-10)
  - Always keep spare nozzles

Problem: Under-Extrusion (Thin Walls, Weak Print) Symptoms:

- Print walls thinner than expected
- Weak layer bonding (layers separate easily)
- Visible horizontal lines in walls
- Light-colored sections in print

Diagnosis Checklist:

- ☐ Nozzle partially clogged?
- ☐ Extrusion multiplier/width set correctly?
- ☐ Hot end temperature too low?
- ☐ Filament diameter inconsistent?
- ☐ Drive gear slipping?

Solutions:

1. Check extruder steps/mm - calibrate e-steps if possible
2. Increase flow rate - try 105-110% in slicer
3. Raise temperature - add 5-10C
4. Slow down print speed - reduce 10-20%
5. Check filament quality - test with different brand/batch
6. Clean drive gear - remove plastic buildup

Problem: Over-Extrusion (Blobs, Rough Texture) Symptoms:

- Excess material squishing out between layers
- Rough, bumpy texture on surface
- Blobs or zits on walls
- Layers slightly translucent

Diagnosis Checklist:

- ☐ Temperature too high?
- ☐ Extrusion width too large?
- ☐ Line width set wider than nozzle?
- ☐ Flow rate too high?

Solutions:

1. Reduce flow rate - try 95% in slicer
2. Lower temperature - reduce 5-10C
3. Check line width - should be ~1.2x nozzle diameter (0.4mm nozzle = 0.48mm width)
4. Speed up print - higher speeds reduce oozing
5. Check nozzle size - confirm you're using correct setting

Problem: Layer Shifting Symptoms:

- Layers offset horizontally mid-print
- X or Y axis suddenly jumps
- Top portion of print misaligned
- Usually happens at specific layer height

Diagnosis Checklist:

- ☐ Mechanical: Check X/Y pulley teeth, belts, screws loose?
- ☐ Collision: Does print hit extruder/frame?
- ☐ Speed: Trying to move too fast, losing steps?
- ☐ Support: Is print unstable/wobbly?
- ☐ Firmware: Recent changes to acceleration settings?

Solutions:

1. Check mechanics:
    - Manually move X/Y freely (no resistance)
    - Tighten all visible screws
    - Check belts for fraying (replace if damaged)
2. Reduce speed:
    - Lower travel speed by 20-30%
    - Reduce acceleration in firmware
    - Slow down print speed by 20%
3. Check print orientation:
    - Rotate model to reduce overhang
    - Add supports to prevent wobbling
    - Ensure solid perimeters around shift point
4. Firmware:
    - Check acceleration settings (try 1000 mm/s)
    - Verify step/mm calibration
    - Update firmware if outdated

Advanced Troubleshooting

Problem: Stringing (Fine Filament Between Prints) Cause: Nozzle oozes while traveling between print sections

Solutions:

1. Reduce temperature by 5-10C
2. Increase retraction distance (0.5-1.5mm more)
3. Increase retraction speed
4. Enable wipe on retract (slicer setting)
5. Check nozzle diameter (worn nozzles ooze more)

Problem: Warping (Corners Curl Up) Cause: Material cools unevenly, contracts differently

Solutions (by material):

- ABS: Heated enclosure, slow cooling, higher bed temp
- PETG: Reduce cooling fan, increase bed temp
- PLA: Usually warps if bed too hot - reduce temperature

Problem: Inconsistent Print Quality Cause: Variable conditions print-to-print

Solutions:

1. Environmental: Maintain consistent room temperature
2. Material: Use same brand/batch filament
3. Calibration: Re-level bed before each print
4. Firmware: Disable bed leveling if unreliable
5. Maintenance: Clean nozzle after each print

Diagnosis Decision Tree

```
Print fails?
+---- No extrusion -> Check nozzle temperature -> Clog? -> Cold pull/poke/replace
+---- Lifts off bed -> Clean bed -> Level bed -> Adjust height -> Increase temp
+---- Breaks mid-print -> Layer shift? -> Check mechanics, speed
| -> Under-extrusion? -> Increase flow/temp/speed
| -> Over-extrusion? -> Decrease flow/temp
+---- Bad surface quality -> Stringing? -> Reduce temp/increase retraction
| -> Blobs? -> Check flow rate, temperature
| -> Warping? -> Adjust bed temp, enclosed space
+---- Print weak -> Under-extrusion -> Increase flow rate, temperature
            -> Poor layer bonding -> Increase bed temp, first layer height
```

Maintenance to Prevent Issues

Issue	Prevention	Frequency
Clogs	Clean nozzle	Before each print
Bed adhesion	Level bed, clean plate	Before each print
Layer shift	Check mechanics, belts	Monthly
Inconsistent quality	Calibrate e-steps	Quarterly
Worn nozzle	Monitor extrusion quality	Every 6 months
Temperature fluctuation	Stable environment	Ongoing

Last Reviewed:
Printer Model:
Notes:

Accessibility Audit - Extension Project

Estimated time: 2-4 hours

Learning Objectives

By completing this project, you will:

- Evaluate digital content for accessibility barriers related to vision, blindness, and low vision
- Apply concrete accessibility best practices (alt-text, code comments, clear language)
- Document improvements and estimate implementation effort
- Strengthen your awareness of inclusive design principles

Objective

- Evaluate one lesson or project for non-visual accessibility and produce a prioritized improvement plan.

Tasks

1. Select one lesson or project (OpenSCAD file + markdown instructions) and review for accessibility shortcomings.
2. Create a short checklist (images alt-text, code walkthroughs, tactile cues, file naming) and apply it to the chosen resource.
3. Implement at least three small improvements (alt text, clearer code comments, or a text-only walkthrough) and document changes.
4. Produce a summary with suggested larger changes and an estimate of the instructor time to implement them.

Deliverable

- A one-page audit report and a patch (diff) suggestion for the target lesson.

Starter files

- starter.scad - small tactile marker scaffold for testing contrast and tactile cues.

Assessment Questions (Optional)

1. What are three key accessibility barriers you discovered in your chosen resource?
2. Which of your three improvements was easiest to implement, and which was most challenging?

3. How could the instructor prioritize large-scale accessibility improvements across the entire course?

Accessibility Audit - Student Documentation Template (Extension Project)

- Author:
- Date:
- Description: Audit 3D printing tools and workflows for accessibility barriers and recommend improvements.

Audit Scope

Tools/workflows to audit:

1. _
2. _
3. _

Testing methodology (screen reader, keyboard-only, other):

Detailed Findings

Tool 1: _

Automated Accessibility Checks

- (Screenshots/results of testing)

Screen Reader Testing

- Navigation: (clear/unclear/broken)
- Output readability: (readable/partially readable/unreadable)
- Error messages: (helpful/unclear/missing)
- Specific barriers found:

Keyboard Navigation

- All functions accessible via keyboard: Yes / No
- Tab order logical: Yes / No
- Specific barriers:

Recommendations for Tool 1

Recommendation	Priority	Feasibility	Impact

Recommendation	Priority	Feasibility	Impact

Tool 2: _ (Repeat structure above)

Tool 3: _ (Repeat structure above)

Summary of Barriers

Barrier	Frequency	Severity	Tools Affected

Recommendations Matrix

Priority	Feasibility	Impact	Recommenda-tion	Implementation Steps
High	Easy	High		
High	Moderate	High		
Moderate	Easy	Medium		

Reflections

- What surprised you about accessibility barriers in these tools?
- How would you prioritize improvements if you had limited resources?
- What role should users with disabilities play in accessibility testing?
- How can accessibility become part of the design culture?

Action Plan

If implementing improvements, describe your plan:

- Which improvements will you tackle first?
- How will you measure success?
- Timeline for implementation:

Attachments

☐ Screenshots of accessibility tests
☐ Screen reader testing notes (with timestamps)
☐ Keyboard navigation checklist

☐ Detailed findings document
☐ Recommendations matrix
☐ Action plan (if implementing)

Teacher Feedback

Category	Score	Notes
Problem & Solution (0-3)		
Design & Code Quality (0-3)		
Documentation (0-3)		
Total (0-9)		

Feedback:

Accessibility Audit - Teacher Template (Extension Project)

Briefing

Students conduct a comprehensive accessibility audit of 3D printing workflows and tools. This project emphasizes universal design principles, accessibility testing, and inclusive documentation.

Key Learning: Accessibility as a design practice; testing with assistive technology; inclusive documentation.

Real-world Connection: Universal design benefits all users. Accessibility is increasingly a legal and ethical requirement in professional contexts.

Constraints

- Audit must cover at least three tools or workflows (editor, slicer, terminal)
- Testing must include both automated checks and user feedback
- Recommendations must be specific and actionable
- Documentation must support future accessibility improvements

Functional Requirements

- Audit identifies specific accessibility barriers with clear descriptions
- Testing conducted with screen reader and keyboard-only navigation
- Recommendations are prioritized by impact and feasibility
- Documentation guides future accessibility improvements

Deliverables

- Completed audit template

- Detailed findings for each tool/workflow
- Screen reader testing log
- Recommendations matrix (priority, feasibility, impact)
- Reflection on accessibility challenges and opportunities
- Action plan for improving course materials

Rubric

Category 1: Problem & Solution (0-3) Audit is thorough and identifies real barriers. Recommendations are actionable.

Category 2: Design & Code Quality (0-3) Testing methodology is rigorous. Findings are specific and well-documented.

Category 3: Documentation (0-3) Audit template complete. Recommendations prioritized. Action plan clear.

Assessment Notes

- Strong submissions: Show rigorous testing methodology, specific barrier descriptions, prioritized recommendations, and sincere reflection on inclusion
- Reinforce: Accessibility is everyone's responsibility; small improvements compound
- Extension: Implementation of recommended improvements; follow-up testing

Lesson 11: Stakeholder-Centric Design and the Beaded Jewelry Project

Estimated time: 90-120 minutes | Level: Advanced

Learning Objectives

- Conduct stakeholder interviews to identify needs, wants, and constraints[247]
- Extract functional requirements from interview data[248]
- Design a product that meets explicit requirements (not assumptions)[249]
- Iterate design based on stakeholder feedback[250]

[247] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[248] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[249] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[250] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/

- Document design decisions for reproducibility and accessibility[251]

Materials

- Interview worksheet (provided below)
- Notebook/laptop for notes
- Access to stakeholder (or use provided scenario)
- 3dMake project scaffold
- Calipers for measurement
- Reference: master-rubric.md for design documentation standards

Extension Project: Complete Beaded_Jewelry_Unit to apply stakeholder-centric design principles in a real project context.

Why This Matters: The Design Thinking Process

Most design failures start before CAD. They happen because:

- Assumption Problem: Designer assumes what users need without asking
- Specification Problem: Unclear requirements lead to wasted iterations
- Accessibility Problem: Features designed without considering all users

This lesson flips the process: Listen First, Design Second [252].

The Design Thinking Pipeline

```
1. Empathize (Listen to stakeholder)
   v
2. Define (Extract functional requirements)
   v
3. Ideate (Sketch/prototype solutions)
   v
4. Prototype (Build in CAD)
   v
5. Test (Get feedback)
   v
6. Iterate (Refine based on feedback)
```

Stakeholder Interviews (20-30 minutes)

What is a Stakeholder?

A stakeholder is anyone who will use, interact with, or be affected by your design. Examples:

- End User: Person who directly uses the product
- Maintenance Person: Someone who repairs/replaces parts
- Budget Holder: Person responsible for cost
- Safety Officer: Person ensuring regulatory compliance

[251] Accessible Design Documentation - https://www.a11y-101.com/design/inclusive-documentation
[252] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking

Open-Ended Interview Questions

Good interview questions:

- [YES] Start with "What," "How," "Why"
- [YES] Don't suggest answers
- [YES] Follow up with "Tell me more..."
- [YES] Listen for numbers, specific stories, pain points

Bad interview questions:

- [NO] "Do you like...?" (Yes/No - no detail)
- [NO] "Would you prefer A or B?" (Forced choice)
- [NO] "Don't you think...?" (Leading question)

Sample Interview Script (Beaded Jewelry)

You're designing a custom beaded bracelet holder. Here's how to interview a stakeholder:

```
Opening:
"Hi [name], I'm designing a way to store and display beaded
bracelets. I'd love to understand how YOU currently use bracelets
so I can design something that works for you. Is now a good time?"
Question 1: "Tell me about your current bracelet collection.
How many do you have? What are they made of?"
Question 2: "How do you currently store or display them?
What works well? What's frustrating?"
Question 3: "If you could design the perfect bracelet holder,
what would it look like? What would it do?"
Question 4: "Are there any constraints I should know about?
(Size of your space, colors, budget, etc.)"
Question 5: "Who else might use this? How would their needs
differ from yours?"
Closing:
"Thank you for your time. I'll design based on what you've told me
and come back with a prototype for feedback."
```

Step-by-Step Interview Task

Duration: 15 minutes total

1. Prepare (5 min):
   - Choose your stakeholder or use the scenario below
   - Write 4-6 questions using the template above
   - Explain the project and get permission to take notes
2. Conduct (10 min):
   - Ask your prepared questions
   - Write down specific details (numbers, names, quotes)
   - Listen more than you talk
   - Ask "Why?" when you hear interesting responses
3. Document (Immediately after):

- Write a 2-3 sentence summary
- Extract 3-5 key needs/constraints
- Identify any questions that still need answers

Scenario: If You Don't Have a Real Stakeholder

Scenario: Meet Sarah, a jewelry enthusiast

Sarah has 15+ beaded bracelets but they're tangled in a drawer. She wants to:

- Display them where she can see them
- Keep them organized by color/style
- Protect them from tangling
- Fit in a small space on her desk

Constraints:

- Budget: $15 materials cost
- Space: 6" x 6" footprint maximum
- Aesthetic: Minimalist, natural wood preferred
- Timeline: Needs it within 2 weeks

Extract Functional Requirements (20-30 minutes)

From Interview to Requirements

A functional requirement is specific, measurable, and testable:

Interview Data	[	NO
"I have 15 bracelets"	Lots of storage	Hold 15-20 bracelets without damage
"They're tangled in my drawer"	Keep organized	Vertical hanging system to prevent tangling
"I want to see them"	Display feature	Visible from above (no lid)
"Minimalist, natural wood"	Nice looking	Wood construction, 2 colors, clean lines

Functional Requirements Template

For your beaded jewelry holder, extract 5-8 requirements:

```
Functional Requirements:
1. Capacity: Hold bracelets
2. Organization: Arrange by (color/size/style)
3. Protection: Prevent (tangling/damage)
4. Display: Visible from (above/front/sides)
5. Dimensions: mm wide, mm deep, mm tall
6. Material: (wood/plastic/metal)
7. Accessibility: (non-visual inspection method)
8. Durability: Withstand (weight/use/time)
```

Measurable Acceptance Criteria

For each requirement, define how you'll TEST it:

Requirement	Measurement Method
Hold 15 bracelets	Count actual bracelets loaded; verify no deformation
Prevent tangling	Manual inspection: no twisting/knotting after 1 week storage
Dimensions	Measure with calipers; compare to design parameters
Wood material	Visual inspection; material weight check
Withstand weight	Load test: 2kg weight for 1 hour; check for deformation

Design the Bracelet Holder (30-40 minutes)

Design Strategy

Based on functional requirements, create a parametric SCAD model:

```
// Parametric Beaded Bracelet Holder
// Designed for: [Stakeholder name]
// Requirements: [List 3-5 key requirements]
// === PARAMETERS (from stakeholder requirements) ===
holderwidth = 150; // mm - width for 15+ bracelets
holderdepth = 80; // mm - depth for stability
holderheight = 120; // mm - height for display
numpegs = 16; // Number of hanging pegs
pegdiameter = 6; // mm - bracelet hang diameter
pegspacing = holderwidth / (numpegs + 1);
// === DESIGN ===
module base() {
  // Wooden-look base plate
  cube([holderwidth, holderdepth, 10]);
}
module backwall() {
  // Back support for stability
  translate([0, 0, 10])
    cube([holderwidth, 5, holderheight]);
}
module hangingpegs() {
  // Pegs for bracelets to hang
  for (i = [1 : numpegs]) {
    xpos = pegspacing * i;
    translate([xpos, holderdepth/2, holderheight - 20])
      cylinder(r=pegdiameter/2, h=30, $fn=16);
  }
}
module braceletholder() {
  // Combine all components
  union() {
    base();
    backwall();
    hangingpegs();
  }
}
// Main
braceletholder();
```

Customization Based on Stakeholder Feedback

Before printing, modify parameters:

```
Interview revealed:
- "15 bracelets is about right"
- "I like minimal spacing"
- "Natural wood color"
- "Small desk space - maybe 5 inches wide?"

Updated parameters:
numpegs = 15; // Exact count for their collection
holderwidth = 127; // 5 inches in mm
pegspacing = holderwidth / (numpegs + 1);
```

Testing Non-Visually

```
# Describe the model to verify it matches requirements
3dm describe src/main.scad

# Expected output should mention:
# - 15 hanging pegs arranged horizontally
# - Base plate for stability
# - Back wall for support
# - Dimensions approximately 127mm x 80mm x 120mm

# Generate preview for tactile inspection
3dm preview src/main.scad
```

Iterate Based on Feedback (20-30 minutes)

Feedback Loop

After printing a test version:

1. Measure physical parts:

```
# Use calipers to verify
- Actual width: mm (design: 127mm)
- Actual peg diameter: mm (design: 6mm)
- Actual spacing: mm
```

2. Test with actual bracelets:

   - Can 15 bracelets actually hang?
   - Do they tangle?
   - Do pegs support the weight?
   - Is it stable on the desk?

3. Document feedback:

```
Test Results:
- [YES] Holds 15 bracelets comfortably
- Pegs slightly too thin (bracelets slip)
- [YES] No tangling observed
- Wobbles slightly - needs wider base

Adjustments for v2:
- Increase pegdiameter: 6mm -> 8mm
- Increase basedepth: 80mm -> 100mm
```

4. Iterate:

```
// Modified parameters for v2
pegdiameter = 8; // Thicker pegs (8mm instead of 6mm)
holderdepth = 100; // Wider base (100mm instead of 80mm)
```

5. Reprint and test:
   - Does v2 address the issues?
   - Any new problems?
   - Is it ready for final design?

Accessible Iteration Tracking

Document all iterations in a spreadsheet or text file:

```
Version | Peg Size | Base Depth | Fits Bracelets | Stable | Notes
--------|----------|------------|----------------|--------|--------------------
v1 | 6mm | 80mm | [YES] (14/15) | | Wobbles, pegs thin
v2 | 8mm | 100mm | [YES] (15/15) | [YES] | Perfect fit, stable
v3 | 8mm | 100mm | [YES] (15/15) | [YES] | With wood stain
```

Complete Project: Beaded Jewelry Bracelet Holder

Project Requirements

Objective: Design a customized bracelet holder based on an actual stakeholder's needs.

Deliverables:

1. Interview Documentation (1-2 pages)
   - 4-6 interview questions asked
   - Stakeholder's 3-5 key needs
   - Extracted functional requirements
   - Acceptance criteria for each requirement
2. Design SCAD File (`braceletholder.scad`)
   - Parametric design with all requirements
   - Clear parameters at top (easily customizable)
   - Well-commented modules
   - Matches stakeholder dimensions
3. Iteration Log (2-3 pages)
   - Version 1 test results
   - Feedback from stakeholder
   - Changes made for v2
   - Final design specifications

4. Printed Prototype (physical)
    - Demonstrates final design
    - Holds actual bracelets from stakeholder
    - Passes all acceptance criteria
5. Final Documentation (1-2 pages)
    - Design process summary
    - Stakeholder feedback quotes
    - Design decisions justified
    - Lessons learned

Assessment Rubric (0-9 scale)

Problem & Solution (0-3 points)

- Stakeholder clearly identified and interviewed
- Requirements extracted from interview (not assumptions)
- Functional requirements are specific and measurable

Design & Code Quality (0-3 points)

- SCAD code is parametric and well-organized
- Design meets all extracted requirements
- Iterations show thoughtful refinement based on feedback

Documentation (0-3 points)

- Interview process clearly documented
- Design decisions justified with stakeholder feedback
- Iteration log shows testing methodology
- Accessible description of design (3dm describe output)

Quiz - Lesson 3dMake.11 (10 questions)

1. What is the primary purpose of conducting stakeholder interviews before design[253]?
2. Give one example of an open-ended interview question and one bad (leading) question[254].
3. What is the difference between a ”need” and a ”want” in the design context[255]?
4. How do you convert interview data into functional requirements[256]?
5. What is an acceptance criterion and why is it important[257]?

[253]Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[254]Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[255]Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[256]Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[257]Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/

6. Describe how you would test a bracelet holder's capacity non-visually using measurements[258].
7. True or False: You can design a good product without talking to the person who will use it.
8. Explain why iteration based on real feedback is better than guessing[259].
9. How would you document design changes in a way that's accessible to non-visual users[260]?
10. What should you do if your prototype fails a stakeholder's test[261]?

Extension Problems (10)

1. Conduct interviews with 3 different stakeholders for the bracelet holder; compare their requirements[262].
2. Design 3 different bracelet holder variants for different stakeholder types (minimalist, maximalist, travel-focused)[263].
3. Create a complete functional requirements document with measurable acceptance criteria[264].
4. Build and test 2 iterations of the bracelet holder; document all changes[265].
5. Develop a design specification document that someone else could follow to recreate your design[266].
6. Conduct a "design review" with your stakeholder; document their feedback and your responses[267].
7. Create a manufacturing specification: tolerances, materials, assembly instructions, quality checks[268].
8. Build a customer feedback tracking system; document what works and what could be improved[269].
9. Design an accessibility checklist for the bracelet holder: non-visual inspection, measurement-based testing, assembly ease[270].
10. Write a comprehensive design case study: from initial interview through final product, including all iterations, feedback, and lessons

[258] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[259] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[260] Accessible Design Documentation - https://www.a11y-101.com/design/inclusive-documentation
[261] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[262] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[263] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[264] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[265] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[266] Accessible Design Documentation - https://www.a11y-101.com/design/inclusive-documentation
[267] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[268] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[269] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[270] Accessible Design Documentation - https://www.a11y-101.com/design/inclusive-documentation

learned[271][272][273].

Summary: From Interview to Design

The 11 lessons have taken you from beginner to advanced designer. This final lesson shows you that the best designs start with listening, not assuming.

The Full Journey

1. Lessons 1-3: Foundation (setup, primitives, parametric code)
2. Lessons 4-5: Verification & Safety (AI checking, physical safety)
3. Lessons 6-7: Applied Projects (keycap, phone stand)
4. Lesson 8: Advanced Design (tolerance, interlocking, complex assemblies)
5. Lesson 9: Automation (batch processing, scripting)
6. Lesson 10: Mastery (troubleshooting, QA, testing)
7. Lesson 11: Leadership (listening to stakeholders, designing for real users)

Next Steps After This Lesson

You're now ready for:

- [YES] Professional product design workflows
- [YES] Customer-driven development
- [YES] Small-batch manufacturing
- [YES] Open-source design contributions
- [YES] Teaching others these skills

References

Stakeholder Interview Template

Structured interview guide for gathering requirements directly from end-users and stakeholders.

Pre-Interview Preparation

Interview Details:

- Date: _
- Time: _
- Location: _
- Stakeholder Name: _
- Role/Title: _

[271] Design Thinking Process - IDEO - https://www.ideo.com/perspectives/design-thinking
[272] Functional Requirements in Product Design - https://www.productschool.com/blog/requirements/
[273] Accessible Design Documentation - https://www.a11y-101.com/design/inclusive-documentation

- Contact: _

Interview Objective: (What do you specifically want to learn?) _

Estimated Duration: 45-60 minutes

Interview Sections

Section 1: Context & Background (5 minutes) Opening Questions:

1. ”Tell me about your role in this project.”
    - _
2. ”What problem are we trying to solve?”
    - _
3. ”Who else will be affected by this design?”
    - _

Section 2: Current Situation (10 minutes) Current Process:

1. ”How do you currently solve this problem?”
    - _
2. ”What works well about the current approach?”
    - _
3. ”What frustrates you most about the current process?”
    - _

Pain Points:

- Primary issue: _
- Secondary issues: _

Section 3: Ideal Solution (15 minutes) Requirements (Ask open-ended):

1. ”Ideally, what would this solution do?”
    - _
2. ”How would you use it?”
    - _
3. ”Where would it be used?”
    - _
4. ”When would it be needed?”

- _

Specific Constraints:

- Size requirements: _
- Weight constraints:
- Temperature tolerance: _
- Durability needs:
- Cost budget:
- Timeline: _

Section 4: Usage Patterns (10 minutes) Frequency & Scale:

1. "How often would this be used?"

   ☐ Daily [ ] Weekly [ ] Monthly [ ] Occasionally

2. "How many people need this?"

   - _

3. "For how long each use?"

   - _

User Profile:

- Physical capabilities: _
- Technical expertise: _
- Accessibility needs: _

Section 5: Success Criteria (10 minutes) What defines success? 1. 2. 3.

How will you measure if this works?

- Metric 1:
- Metric 2:

What would be a failure?

Section 6: Preferences & Priorities (10 minutes) Priority Ranking (Ask: "If you could only have 3 features, which would they be?")

1. (Priority: /10)
2. (Priority: /10)
3. (Priority: /10)

Style/Aesthetic:

- Preferred look:
- Must fit with existing: _
- Any visual requirements: _

Material/Finish:

- Preferred materials:
- Texture preference: _
- Color preferences:

Follow-Up Questions

Dig deeper on key points:

- ”Tell me more about that…”
- ”Why is that important?”
- ”What would happen if that weren't included?”
- ”Can you give me an example?”
- ”How would you compare that to…?”

Accessibility Considerations

Always Ask:

- ”Are there any accessibility requirements?”
- ”Will this be used by people with different abilities?”
- ”Are there sensory, mobility, or cognitive considerations?”
- ”What assistive devices might be used with this?”

Document:

- Visual considerations: _
- Motor/dexterity needs:
- Hearing/audio considerations: _
- Cognitive/learning preferences:

Post-Interview Notes

Key Takeaways: 1. 2. 3.

Action Items:

- [ ]
- [ ]

Questions to Follow Up On: 1. 2.

Conflicts/Concerns:

Next Steps:

Interview Analysis

After conducting multiple interviews, summarize:

Common Themes (What requirements appeared in multiple interviews?) 1. 2. 3.

Conflicting Requirements (What requirements contradicted each other?)

- Stakeholder 1 wants: _
- Stakeholder 2 wants: _
- Resolution:

Risk Areas

(What might be difficult or risky about this project?)

-

Interview Conducted By: _ Date Documented: _ Approved By Stakeholder: _ (signature/name)

Functional Requirements Template

Document and organize functional requirements derived from stakeholder interviews and research.

Project Information

- Project Name: _
- Version: _
- Date Created: _
- Last Updated: _
- Prepared By: _
- Approved By: _

Executive Summary

Project Description: (One paragraph overview of what this project is about) _ _

Business/User Need: (Why is this project important?) _

Stakeholders:

- Primary user: _
- Secondary users:
- Decision makers:

Functional Requirements

Organization Strategy Requirements organized by:

- ☐ User role/perspective
- ☐ Feature/component
- ☐ Priority level
- ☐ Other: _

Primary Functional Requirements

Core features that define the product

FR1: [Feature Name] Priority: [ ] Critical [ ] High [ ] Medium [ ] Low

Description: What the product must do: _

User Stories:

- "As a [user type], I want to [action] so that [benefit]"
  - _
- "As a [user type], I want to [action] so that [benefit]"
  - _

Acceptance Criteria:

- ☐
- ☐
- ☐

Related Requirements:

FR2: [Feature Name] Priority: [ ] Critical [ ] High [ ] Medium [ ] Low

Description: _

User Stories:

-

Acceptance Criteria:

- ☐
- ☐

Related Requirements:

FR3: [Feature Name] (Continue with additional primary requirements)

Secondary Functional Requirements

Important supporting features

FR-S1: [Feature Name] Priority: [ ] High [ ] Medium [ ] Low

Description: _

Acceptance Criteria:

- ☐
- ☐

FR-S2: [Feature Name] (Continue with secondary requirements)

Non-Functional Requirements

Performance, reliability, and design properties

Performance Requirements

- Response time: _
- Throughput: _
- Resource usage: _

Reliability Requirements

- Failure rate acceptable: _
- Recovery capability: _
- Data persistence: _

Physical Requirements (for 3D printed objects)

- Dimensions:
- Weight capacity:
- Material properties:
- Temperature range:
- Durability:

Accessibility Requirements

- ☐ Usable by people with visual impairment
    - How:
- ☐ Usable by people with motor impairment
    - How:
- ☐ Usable by people with hearing impairment

- How:

- ☐ Usable by people with cognitive differences
    - How:

Constraint Requirements

Limitations on design and implementation

Technical Constraints

- Must work with:
- Must not require:
- Must be compatible with:

Physical Constraints

- Cannot exceed (size/weight/cost): _
- Must fit in/with: _
- Must be available by (date): _

Regulatory/Safety Constraints

- Must comply with:
- Must not:
- Safety considerations: _

Cost Constraints

- Maximum budget:
- Target unit cost: _

Environmental Context

Use Environment

- Location(s): _
- Climate conditions: _
- Physical surroundings: _
- Typical usage pattern: _

Maintenance & Lifecycle

- Expected lifespan:
- Maintenance needed:
- End-of-life handling:

Dependency Mapping

Requirements that depend on other requirements:

Requirement	Depends On	Notes
FR1		
FR2	FR1	Cannot implement without FR1
FR3		

External dependencies:

- Third-party components needed:
- Integration points:

Scope Definition

What IS In Scope

-
-
-

What IS NOT In Scope

-
-

Future Considerations (Out of Scope but noted)

-

Change Control

Requirements Changes:

Change Request	Date	Reason	Status

Verification Plan

How will we verify each requirement is met?

Requirement	Verification Method	Test Case	Status
FR1	[Inspection/Test/Demo]		
FR2	[Inspection/Test/Demo]		

Sign-Off

Stakeholder Approval:

Stakeholder	Title	Signature	Date

Requirements Baseline Approved: Date: Status: Approved / Pending / Rejected

Reference Documents

- Interview notes:
- Design sketches:
- Related specifications: _
- Standards/guidelines:

Document Version History:

Version	Date	Author	Changes
1.0			Initial draft

Design Specification Template

Technical design document specifying how requirements will be implemented.

Design Document Information

- Project: _
- Design Version: _
- Date: _
- Designed By: _
- Reviewed By: _

Design Overview

Problem Statement: (What are we solving?) _

Solution Approach: (High-level how we'll solve it) _

Key Design Decisions: 1. 2. 3.

Component Breakdown

Component 1: [Component Name] Purpose: _

Specifications:

- Dimensions:
- Material:
- Quantity:

Design Rationale: (Why this approach?) _

Related Components:

- Connects to: _
- Interfaces with:

Manufacturing Considerations:

- Support structures needed: _
- Orientation for print: _
- Estimated print time: _

Component 2: [Component Name] (Repeat structure for each component)

Assembly Design

Overall Assembly Structure: (How components fit together) _

Assembly Sequence: 1. 2. 3.

Fastening Methods:

☐ Snap fits (location: _)
☐ Threaded inserts (quantity:)
☐ Glue/adhesive (type:)
☐ Other: _

Assembly Challenges & Solutions:

- Challenge: _ Solution:

Material Selection

Primary Material:

Why this material?

Property	Requirement	Selected Material	Alternative
Strength			
Flexibility			
Temperature			
Cost			
Availability			

Material Properties:

- Nozzle temperature:
- Bed temperature:
- Print speed: _
- Support required:
- Post-processing:

Material Alternatives & Trade-offs:

- Option 1: _
- Option 2: _
- Selected option rationale: _

Design Features

Feature 1: [Feature Name] Purpose: _

Design Details:

- Dimensions:
- Placement: _
- Tolerances:

Rationale: (Why designed this way?) _

Related Requirement: (Which requirement does this fulfill?) _

Feature 2: [Feature Name] (Repeat for each significant feature)

Tolerance & Fit Analysis

Critical Dimensions:

Dimension	Tolerance	Rationale	Risk
	+/-_mm		
	+/-_mm		

Fit Relationships:

- Part A to Part B: _
- Part B to Part C: _

Validation Plan:

☐ Print test version before production
☐ Measure critical dimensions
☐ Test assembly fit
☐ Perform functional test

Manufacturing Planning

Print Parameters Slicer Settings:

- Layer height:
- Infill:
- Support material:
- Raft/brim: _

Print Time & Material:

- Estimated time:
- Material weight: _
- Cost estimate: _

Orientation Strategy: (How will this be oriented during printing?)

- Rationale: _
- Risks: _
- Mitigation:

Post-Processing Cleanup:

☐ Remove supports (method:)
☐ Remove raft/brim (method: _)
☐ Sand edges (grit: _)
☐ Smooth surfaces (method:)

Finishing:

☐ Paint/coat (type:)
☐ Treat for durability:
☐ Inspect quality:

Assembly Work:

- ☐ Insert threaded nuts/inserts
- ☐ Assemble components
- ☐ Test functionality

Estimated Total Time: hours

Testing Plan

Functionality Tests:

Test	Method	Pass Criteria	Status

Durability Tests:

- Stress test: _
- Environmental exposure:
- Lifecycle test: _

Tolerance Verification:

- Measure key dimensions
- Compare to specification
- Document variance

Design Risks & Mitigation

Risk 1:

- Description: _
- Probability: [ ] High [ ] Medium [ ] Low
- Impact: [ ] Critical [ ] Major [ ] Minor
- Mitigation:
- Contingency: _

Risk 2: (Continue for identified risks)

Design Alternatives Considered

Alternative 1:

- Approach: _
- Pros:
- Cons:
- Why not selected:

Alternative 2: (Continue for each alternative considered)

Technical Specifications

Dimensional Drawing Reference (Attach or describe detailed drawings/models)

- CAD file:
- Drawing revision:

Performance Specifications

- Load capacity: _
- Temperature range: _
- Lifespan:
- Accuracy/precision:

Bill of Materials (BOM)

Item	Part #	Quantity	Unit Cost	Notes
3D Printed Main Body	-	1	$0.XX	PLA

Total Material Cost: $

Design Verification Checklist

Before manufacturing:

- ☐ All components specified
- ☐ Material selected and justified
- ☐ Assembly method documented
- ☐ Tolerances verified
- ☐ Manufacturing plan complete
- ☐ Testing approach defined
- ☐ Risks documented
- ☐ Cost estimates calculated
- ☐ Timeline realistic
- ☐ Design approved by stakeholders

Design Change Log

Change #	Date	Description	Reason	Impact

Sign-Off

Design Approval:

Role	Name	Signature	Date
Designer			
Reviewer			
Stakeholder			

Status:

- ☐ Approved - Ready for manufacturing
- ☐ Approved with conditions:
- ☐ Not approved - Revisions needed:

Appendices

- Appendix A: Detailed CAD Models
- Appendix B: Assembly Instructions
- Appendix C: Material Data Sheets
- Appendix D: Test Reports

Feedback Collection Template

Structured approach to gathering and analyzing feedback from users and stakeholders throughout the design process.

Feedback Collection Strategy

Collection Points

- ☐ Prototype testing (early design)
- ☐ Mid-development review
- ☐ Beta testing (near completion)
- ☐ Post-launch review
- ☐ Long-term usage feedback

Target Audience

- Primary users:
- Secondary users:
- Stakeholders:

Prototype Feedback Form

Feedback Session Information

- Feedback Date:
- Tester Name:
- Tester Role:
- Prototype Version:
- Session Duration:

Section 1: First Impressions (5 minutes)

Initial Reactions

1.

   "What is your first impression of this design?"

2. "Is it what you expected?"
   - ☐ More than expected [ ] As expected [ ] Less than expected
   - Explanation:
3. "Would you use this?"
   - ☐ Definitely [ ] Maybe [ ] No [ ] Not sure
   - Why?

Section 2: Usability (15 minutes)

Using the Product

1. "Can you show me how you would use this?"
   - (Observe: difficulties, hesitations, confusion)
   - Notes:
2. "Was it easy to understand how to use?"
   - ☐ Very easy [ ] Easy [ ] Neutral [ ] Difficult [ ] Very difficult
   - What was confusing?
3.

   "Are there any parts that don't work well together?"

Specific Usability Issues

Issue	Severity	Suggestion
	High/Med/Low	
	High/Med/Low	

Section 3: Functionality Assessment (10 minutes)

Does it do what it's supposed to?

Feature/Function	Works Well	Works OK	Doesn't Work	Needed But Missing
				-
				-
				-

Critical Functions:

- Most important for your use:
- Missing capability:

Section 4: Physical Design Assessment (10 minutes)

Appearance & Feel

Visual Design

- ☐ Appealing [ ] Acceptable [ ] Unappealing [ ] Doesn't matter
- Suggestions:

Size & Comfort

- Too large / Just right / Too small (circle one)
- Weight: Too heavy / Just right / Too light (circle one)
- Comfort feedback:

Material & Texture:

- Appropriate: [ ] Yes [ ] No [ ] Uncertain
- Feel:
- Durability concern:

Color/Finish:

- Appropriate: [ ] Yes [ ] No
- Preference:

Section 5: Performance & Reliability (10 minutes) During Testing:

1. ”Did it work consistently?”
   - ☐ Always [ ] Usually [ ] Sometimes [ ] Never
   - Issues experienced:
2. ”How solid/durable does it feel?”
   - ☐ Very durable [ ] Durable [ ] Adequate [ ] Fragile [ ] Very fragile
   - Concerns:
3. ”Any parts that seem weak?”

Section 6: Overall Satisfaction (5 minutes) Scoring Matrix:

Criterion	Rating	Comments
Overall satisfaction	1 2 3 4 5	
Likely to recommend	1 2 3 4 5	
Meets my needs	1 2 3 4 5	
Design quality	1 2 3 4 5	
Value for cost	1 2 3 4 5	

(1=Poor, 5=Excellent)

Section 7: Suggestions & Improvements (10 minutes) Open-Ended Feedback:

1.

   ”What would make this better?”

2.

   ”What should we change?”

3.

   ”What should we keep as-is?”

4. ”Would you buy/use this?”
   - ☐ Yes [ ] Maybe [ ] No
   - Why?

Suggestions Priority:

Suggestion	Priority	Category	Effort
	High/Med/Low	Design/Function/Material	Easy/Med/Hard
	High/Med/Low	Design/Function/Material	Easy/Med/Hard

Section 8: Accessibility Check (5 minutes) If applicable to your product:

1. ”Can someone with limited vision use this?”

   ☐ Yes [ ] With difficulty [ ] No
   - How:

2. ”Can someone with limited mobility use this?”

   ☐ Yes [ ] With difficulty [ ] No
   - How:

3.

   ”Are there any accessibility barriers?”

Post-Testing Notes

Tester Behavior Observations: (What did you notice about how they used it?)

Surprises: (Anything unexpected?)

Key Insights:

1.
2.

Critical Issues Found:

-
-

Non-Critical Issues:

-

Feedback Analysis (After Multiple Sessions)

Common Themes Frequently Mentioned Issues:

1. (Mentioned by of% testers)
2. (Mentioned by of%)
3. (Mentioned by of%)

Frequently Praised Aspects:

1.
2.

Data Summary

Metric	Average	Range	Status
Overall Satisfaction	/5		
Likelihood to Recommend	/5		
Meets Needs	/5		
Usability Rating	/5		

Priority Action Items Must Fix (High Priority - blocks usage):

1.
   - Impact if not fixed:
2.
   - Impact if not fixed:

Should Fix (Medium Priority - improves experience):

1.
2.

Nice to Have (Low Priority - enhancements):

1.
2.

Design Changes Resulting from Feedback

Original Design	Feedback	Change Made	Rationale

Follow-Up Actions

Immediate Actions:

- ☐ Fix critical issues:
- ☐ Notify stakeholders:
- ☐ Update design document:

Next Test Cycle:

- ☐ Schedule follow-up testing
- ☐ Prepare updated prototype
- ☐ Revise feedback form if needed

Timeline for Changes:

- Critical fixes by:
- Medium priority by:
- Low priority by:

Stakeholder Communication

Share findings with:

- ☐ Design team
- ☐ Stakeholders
- ☐ User community
- ☐ Management

Format for sharing:

- ☐ Summary report
- ☐ Presentation
- ☐ Video highlights
- ☐ Detailed documentation

Feedback Tracking

Overall Feedback Summary:

Test Round	Date	# Testers	Key Finding	Action Taken
Prototype V1				
Prototype V2				
Beta				

Sign-Off

Feedback Session Conducted By: Analysis Completed By: Reviewed By:

Status:

- ☐ Feedback incorporated into design
- ☐ Partial incorporation (note:)
- ☐ Under review (decision pending)
- ☐ Declined (reason:)

Session Documentation Date: Feedback Database Location:

Master Project Rubric - 0-9 Point Scale

This rubric applies to all Unit 1, 2, and 3 projects unless otherwise noted in the project briefing. Project 0 is complete/incomplete only.

Rubric Overview

Projects are scored from 0-9 across three categories worth 3 points each:

Category	Max Points	What Is Evaluated
Problem & Solution	3	How well the prototype meets the identified problem or functional requirements
Design Quality	3	Complexity, originality, technical quality of the design; evidence of iteration
Documentation	3	Completeness, accuracy, and thoughtfulness of all written deliverables

Category 1: Problem & Solution (0-3 points)

Score	Description
3	The prototype clearly and effectively solves the stated problem. All functional requirements are met. The solution shows evidence of testing against the requirements.
2	The prototype mostly meets the problem. Most functional requirements are met. Minor gaps between the design and the requirements.
1	The prototype partially addresses the problem. Several functional requirements are not met or were not clearly tested.
0	The prototype does not address the stated problem, or no functional requirements were established.

Category 2: Design Quality (0-3 points)

Score	Description
3	The design is original, well-considered, and technically executed. Code (if applicable) is clean, uses variables/modules appropriately, and is well-commented. The print is clean and well-made. Evidence of at least one meaningful revision or iteration.
2	The design is functional and shows thought. Code works but may lack structure (few comments, raw numbers instead of variables). Print quality is acceptable. Some iteration evident.
1	The design is basic or primarily borrowed from another source without modification. Code has issues. Print quality has significant defects that were not addressed.
0	No meaningful original design. Print is not functional or was not completed.

Category 3: Documentation (0-3 points)

Score	Description
3	All required sections are present, complete, and specific. Reflections are thoughtful and reference specific decisions, problems encountered, and learning. Photos are included. Measurements are recorded.
2	Most required sections are present. Some sections are vague or missing detail. Reflections show some thought but are brief or generic.
1	Documentation is incomplete. Major sections are missing or consist of one-line responses. Reflections are minimal.
0	Documentation is not submitted or is essentially empty.

Score Interpretation

Total Score	Interpretation	Next Step
8-9	Excellent work	Move on to next project
6-7	Good work with room for improvement	Move on; instructor may suggest revisiting one element
4-5	Meets basic expectations	Resubmission of specific weak areas recommended
2-3	Does not meet expectations	Resubmission required
0-1	Missing major deliverables	Meet with instructor; create a completion plan

Resubmission Policy

Students may resubmit any project as many times as they need to improve their score. Resubmissions must include:

1. A one-paragraph explanation of what was changed and why

The resubmission score replaces the original score.

Project 3: Beaded Jewelry - OpenSCAD Design

Accessibility: When including images or diagrams, add short alt-text and provide a comment-based walkthrough for any `.scad` examples so screen-reader users can follow the design steps.

Unit: 3 - Open-Ended Projects Estimated Duration: 1 week (self-paced: plan milestones below) Deliverables due: End of week (see milestones)

Project Brief

Design and produce a wearable beaded jewelry piece that includes at least eight 3D-printed beads generated in OpenSCAD. Your design must use two distinct parametric bead modules and combine them into a completed, wearable piece (necklace, bracelet, or similar).

Constraints (must follow)

- Your prototype must include a 3D-printed component designed in OpenSCAD.
- Code your project in a single `.scad` file that parameterizes bead shapes for repeatability.
- Use at least two different bead shapes in the final assembly.
- The final piece must be wearable and assembled - not just a set of loose beads.

Learning Objectives

- Create parametric OpenSCAD modules for repeated geometry
- Combine modular parts into a coherent assembled object
- Document design decisions and printing notes for reproducibility
- Evaluate designs against measurable functional requirements

StepbyStep Milestones

1. Project setup (Day 1)

- Read this briefing and the Unit 3 lessons.
- Create a folder for your project and initialize a short Design Notes document.

2. Bead module development (Days 1-2)

- Implement `bead_A(size, detail)` and `bead_B(size, detail)` parametric modules.
- Test-print a single bead from each module; record optimal print temperature/bed settings.

3. Assembly and iteration (Days 3-4)

- Create an assembly script that arranges at least 8 beads and tests fit/tolerances.
- Iterate bead hole diameter and test-strung spacing until beads slide but do not fall off.

4. Final prototype & documentation (Day 5)

- Print final beads, assemble the piece, photograph the result, and prepare the deliverables.

Deliverables

Submit both digitally and physically as instructed:

- `.scad` file containing parametric bead modules and the assembly script
- `.stl` files (exported as needed) for printed parts
- Technical documentation (Google Drive link or similar) including:
  - Design Notes (ideas, measurements, param values)
  - Construction / 3D printing notes (temperatures, speeds, supports)
  - Photos of final prototype (multiple views)
- Physical turn-in: one assembled piece (if required by instructor)

Functional Requirements (examples - adapt to your design)

- The bead module must allow a hole diameter adjustable in 0.1 mm increments between 2.0-4.0 mm.
- The assembly must include at least 8 beads and remain wearable (fits comfortably on intended body part).
- The OpenSCAD file must be parameterized such that changing a single `scale` parameter adjusts bead size consistently.
- The assembled piece must not have sharp edges that would injure skin under normal use.

Grading Rubric (simplified, 0-9 scale)

- Implementation & parametric code: 3 points
- Functionality & wearability: 3 points
- Documentation & print notes: 2 points
- Presentation & photos: 1 point

Quiz - Project 3: Beaded Jewelry (10 questions)

1. Why use parametric modules for repeated parts? (one sentence)
2. What is a reliable way to test a bead hole for a given cord diameter?
3. Name two OpenSCAD functions or techniques useful for repeating geometry.
4. What printing setting is most likely to affect hole diameter accuracy?
5. How would you document an iteration that changed hole diameter from 2.5 mm to 2.7 mm?
6. True/False: Once your first bead prints successfully, all subsequent iterations will print exactly the same. (Answer: False - each print can vary due to temperature, humidity, and material batch differences)
7. Short answer: Explain what ”wearability” means for a jewelry design. What are two specific checks you would perform to confirm a necklace is comfortable to wear?

8. Practical scenario: Your bead design uses a 3 mm hole diameter. When you string it on a cord, it's too tight and won't slide. What are two possible solutions you could implement in OpenSCAD to fix this?
9. Multiple choice: When documenting your design for reproducibility, what should you record? (A) Final bead dimensions only (B) All design iterations, parameter changes, and failed attempts (C) Just the successful version - Answer: B
10. Reflection: Describe how the iterative design process (design -> print -> test -> modify) applies specifically to creating a functional wearable. Why is rapid testing and documentation critical for jewelry design?

Answer key (instructor use):

1. To allow repeatable, adjustable geometry and quick global changes.
2. Print a small tolerance test piece and measure fit; record results.
3. `for()` loops, `module` parameters, `translate()` and `rotate()`; `scale()`.
4. Nozzle diameter and extrusion multiplier; also bridging/cooling.
5. Note the parameter change, reproduce the print settings, measure and record fit, update Design Notes.

Extension Problems (apply project skills)

1. Design an interlocking bead (snap-fit) and describe the tolerances required.
2. Create a parametric clasp module that integrates with your bead string and documents a pass/fail test.
3. Modify one bead to include decorative text or pattern generated procedurally in OpenSCAD.
4. Convert your single `.scad` project into a small library: separate bead modules into include files and demonstrate reuse.
5. Propose a modification to make the piece weather-resistant for outdoor wear (materials, coatings, or geometry).
6. Design and print a complete jewelry set: matching beads, clasp, and string connector with consistent design language.
7. Create a variant generator that produces 10+ different bead designs with single-parameter changes; document aesthetic and functional differences.
8. Build a design system document for your jewelry: modular bead library, material requirements, assembly instructions, and care guide.
9. Investigate material effects: print the same bead in 2+ materials; compare durability, aesthetic, and wearability.
10. Develop a parametric customization guide: enable users to modify size, spacing, color (if multi-material), and aesthetics through top-level variables.

Submission Instructions

Upload your digital deliverables to the course Drive folder and email the instructor with the link; bring the printed prototype to class on the due date.

Accessibility

Provide alt-text for photos and a short written walkthrough of how your `.scad` file generates the bead shapes so screen-reader users can understand the sequence and parameters.

Notes

The attached asset `project_3_briefing.txt` was used as the source. If you want me to also copy the original `.txt` into a `.bak` in the same folder, I can do that - tell me and I'll create an adjacent `.bak` file.

3dMake Foundation Final Exam

Name: | Date:

Total Points: 100 (4 points per problem)

Instructions:

- Answer all 25 questions
- For code errors, identify the specific problem and explain why it's wrong
- For behavioral questions, show your reasoning
- For design problems, explain your approach and why it solves the stated challenge
- You may reference the OpenSCAD documentation and appendices
- All work must be your own

Section 1: Error Detection & Code Analysis (Problems 1-10)

For each code block, identify any errors. If there is an error, explain what's wrong and how to fix it. If the code is correct, state "No error" and explain what the code does.

Problem 1: Primitive Definition Error

```
cube([10, 10, 10], center=true);
sphere(r=5, $fn=16);
cylinder(h=20, r=8);
```

Question: Is there an error in this code? If yes, identify it. If no, explain what this code renders.

Answer:

Problem 2: Transform Syntax Error

```
translate([5, 5, 0])
  rotate([0, 45, 0])
    cube([10, 10, 10], center=true);
```

Question: Does this code have a syntax error? Explain what this code does.

Answer:

Problem 3: CSG Operation Error

```
cube([20, 20, 20], center=true);
difference() {
  sphere(r=12);
  cylinder(h=30, r=5, center=true);
}
```

Question: What is wrong with this CSG operation? Explain the fix.

Answer:

Problem 4: Module Definition Error

```
module bracket(width, height, depth) {
  cube([width, height, depth], center=true);
  translate([width/2 + 2, 0, 0])
    cube([4, height, depth/2], center=true);
}
bracket(20, 15, 10);
```

Question: Is there an error in this module definition or call? Why or why not?

Answer:

Problem 5: Parameter & Variable Scoping Error

```
wallthickness = 2;
module hollowcube(size) {
  difference() {
    cube([size, size, size], center=true);
    cube([size - wallthickness, size - wallthickness, size - wallthickness], →
         ↪ center=true);
  }
}
hollowcube(20);
```

Question: Will this code work correctly? If not, what is the problem and how would you fix it?

Answer:

Problem 6: Loop & Iteration Error

```
for (i = [0:5:20]) {
  translate([i, 0, 0])
    cube([4, 4, 4]);
}
```

Question: Will this code produce 5 cubes? Show the positions and explain why or why not.

Answer:

Problem 7: Center Parameter Misunderstanding

```
cube([10, 10, 20], center=false);
sphere(r=5);
```

Question: What is the relationship between these two shapes? Where would the sphere appear relative to the cube?

Answer:

Problem 8: Intersection Error

```
intersection() {
  cube([20, 20, 20], center=true);
  sphere(r=8, $fn=32);
}
```

Question: Is there an error? What will this code render?

Answer:

Problem 9: Nested Transform Error

```
translate([10, 0, 0])
  rotate([0, 0, 45])
    translate([5, 0, 0])
      cube([5, 5, 5], center=true);
```

Question: Are the transforms applied in the correct order? Trace the final position of the cube.

Answer:

Problem 10: Resolution Parameter Error

```
sphere(r=10, $fn=4);
cylinder(h=20, r=8, $fn=3);
cube([10, 10, 10]);
```

Question: Identify the problem(s) with resolution in this code. What will happen when rendered?

Answer:

Section 2: Code Behavior & Theory (Problems 11-17)

For each question, show your reasoning. You may draw diagrams if helpful.

Problem 11: Vertex Coordinates

Question: A cube is defined as `cube([10, 10, 20], center=false)`.

a) List the XYZ coordinates of all 8 vertices.

Answer:

- Vertex 1:
- Vertex 2:
- Vertex 3:
- Vertex 4:
- Vertex 5:
- Vertex 6:
- Vertex 7:
- Vertex 8:

b) Now define the SAME cube with `center=true`. List the NEW coordinates of all 8 vertices.

Answer:

- Vertex 1:
- Vertex 2:
- Vertex 3:
- Vertex 4:
- Vertex 5:
- Vertex 6:
- Vertex 7:
- Vertex 8:

Problem 12: Sphere Geometry

Question: Explain the difference between `sphere(r=10, $fn=8)` and `sphere(r=10, $fn=128)`.

Which would you use for a prototype and which for final printing? Why?

Answer:

Problem 13: Transform Order

Question: Given this code:

```
translate([10, 0, 0])
  rotate([0, 0, 45])
    cube([5, 5, 5], center=true);
```

Does the order matter? What if you swap translate and rotate? Show both final positions.

Answer:

Problem 14: Boolean Operation Behavior

Question: You have a solid cube and you want to create a hole through it. Which CSG operation would you use: `union()`, `difference()`, or `intersection()`?

Explain your choice and write pseudocode showing how you'd accomplish this.

Answer:

Problem 15: Parametric Design Advantage

Question: Compare these two approaches:

Approach A: Hard-coded cube with fixed dimensions

```
cube([10, 10, 20]);
```

Approach B: Parametric cube

```
module parametricbox(width, height, depth) {
  cube([width, height, depth], center=true);
}
parametricbox(10, 10, 20);
```

Why is Approach B better for design iteration? Give an example of how you'd use it.

Answer:

Problem 16: Scale Transform Behavior

Question: If you apply `scale([2, 1, 0.5])` to a `cube([10, 10, 10], center=true)`, what are the NEW dimensions of the cube?

Answer: New dimensions:

Show your calculation:

Problem 17: Library Organization

Question: You've created three useful modules:

- `bracket(width, height, depth)`

- `hollowcube(size, wallthickness)`
- `connectorpin(diameter, height)`

How would you organize these into a reusable library? What file structure would you create and why?

Answer:

Section 3: Design & Problem-Solving (Problems 18-25)

These problems test your ability to design solutions, debug real-world issues, and think creatively.

Problem 18: Tolerance Design Challenge

Question: You're designing a snap-fit connector. The male part has a thickness of 2mm. The female slot needs to accommodate this part with enough flexibility to snap but not fall out.

Should the slot be: a) Exactly 2mm wide b) 2.1mm wide c) 1.9mm wide d) 2.5mm wide

Explain your choice and the design thinking behind it.

Answer:

Problem 19: Design Iteration Problem

Question: You print a keycap with `keysize=12` and the text embossing is too shallow to feel. Your code uses:

```
linearextrude(height=1)
  text("A", size=8);
```

What parameter(s) would you adjust to make the embossing deeper? Show your new code.

Answer:

Problem 20: Error Diagnosis

Question: Your 3dMake build fails with this error: "Geometry is non-manifold." You have this code:

```
difference() {
  cube([20, 20, 20], center=true);
  cylinder(h=30, r=4, center=true);
}
```

Why might this fail? What's the common fix for non-manifold geometry?

Answer:

Problem 21: Multi-Part Assembly

Question: You're designing a two-part box (lid + base). The base has dimensions [50, 30, 20]. The lid should sit on top of the base.

Write parametric modules for both parts and show how you'd position them together. Include appropriate positioning logic.

Answer:

```
module base(length, width, height) {
  // Your code here
}
module lid(length, width, height) {
  // Your code here
}
// Positioning code here:
```

Problem 22: Optimization Challenge

Question: You have a design that takes 5 minutes to render. You notice you have:

```
sphere(r=10, $fn=256);
cylinder(h=20, r=8, $fn=256);
cube([20, 20, 20]);
```

Which parameter(s) would you reduce to speed up rendering while maintaining acceptable quality for a prototype? Explain your choices.

Answer:

Problem 23: Real-World Constraint Problem

Question: A stakeholder requests a custom handle for a tool. They specify:

- Must fit a hand (approximately 80mm long)
- Must accommodate fingers 60mm long inside
- Wall thickness must be at least 3mm for durability
- Should be ergonomic (slightly curved)

Sketch or describe a parametric design for this handle. What parameters would you expose to allow customization?

Answer:

Problem 24: Code Reusability Challenge

Question: You've created a single keycap module. Now you need to create a keyboard with 5 keys arranged in a row. Keys are spaced 15mm apart.

Write code using a loop that creates 5 keycaps with letters A-E, properly spaced.

Answer:

```
module keycap(letter, keysize=10) {
  // Keycap code here (you can assume this exists)
}
// Your loop code here:
```

Problem 25: Design Thinking & Iteration

Question: You've printed Iteration 1 of a product and measured the results. The wall thickness is 3mm but feels too fragile. In Iteration 2, you increased it to 5mm, and now it feels too rigid and won't flex as intended.

For Iteration 3, what thickness would you try and why? How would you make this decision more scientific/data-driven?

Answer:

Bonus Challenge (Optional, +5 points)

Question: Design a parametric model for a custom assistive technology device (e.g., a tactile measuring tool, a custom gripper, an adapted eating utensil, etc.).

- Identify the user's specific need
- Specify the key dimensions and parameters
- Write at least one module with realistic dimensions
- Explain how the design would be tested and iterated

Answer:

```
// Your design here:
```

User Need:

Parameters:

Testing Plan:

Scoring Rubric

Points per Problem	Criteria
4	Correct answer with clear, complete explanation; demonstrates deep understanding
3	Mostly correct answer; minor gaps in explanation or reasoning
2	Partially correct; shows some understanding but has significant gaps
1	Minimal effort; shows limited understanding
0	No answer or completely incorrect

Total Possible: 100 + 5 bonus = 105 points

References You May Use

- 3dMake Quick Reference
- OpenSCAD Cheat Sheet
- Appendix A: Comprehensive Slicing Guide
- Appendix C: Tolerance Testing & QA Matrix
- Appendix D: PowerShell Integration

End of Final Exam

Submission Instructions:

1. Answer all 25 questions completely
2. Show your work for calculations and reasoning
3. Include code samples where requested
4. Submit as a PDF with your name and date
5. Scoring will be based on correctness, clarity, and depth of understanding

Good luck! [celebration]

3dMake Foundation Quick Reference Guide

Fast lookup for lessons, projects, resources, and common tasks

Lesson Quick Reference

All 11 Lessons at a Glance

#	Title	Duration	Level	Main Topics	Key Project
1	Environmental Configuration	60-90m	Beginner	Setup, project structure, `3dm build`	None
2	Geometric Primitives & CSG	60m	Beginner	Primitives, CSG operations, debugging	None
3	Parametric Architecture	60m	Beginner+	Modules, libraries, parameters	None
4	AI Verification	45-60m	Intermediate	`3dm info`, validation, design documentation	None
5	Safety & Physical Interface	60-90m	Intermediate	Safety protocols, materials, pre-print checks	None

#	Title	Duration	Level	Main Topics	Key Project
6	3dm Commands & Text	60-90m	Intermediate	`3dm describe/preview/orient/slice`, embossing	Keycap
7	Parametric Transforms	75-90m	Intermediate+	Transforms, multi-part design, assembly	Phone Stand
8	Advanced Parametric Design	90-120m	Advanced	Tolerance, interlocking features, snap-fits	Stackable Bins
9	Automation & Workflows	60-90m	Advanced	PowerShell scripting, batch processing, CI/CD	[key] Batch Automation
10	Troubleshooting & Mastery	120-150m	Advanced	Measurement, QA testing, diagnostics	[dice] QA + Audit
11	Stakeholder-Centric Design	90-120m	Advanced+	Design thinking, user research, iteration	[beads] Jewelry Holder

4 Reference Appendices

Quick links to comprehensive reference materials:

Appendix	Focus	Size	Use When
A: Comprehensive Slicing Guide	7 major slicers	1,500+ lines	Slicing questions, slicer reference
B: Material Properties	6 materials	1,200+ lines	Choosing material, troubleshooting prints
C: Tolerance & QA	Measurement procedures	1,200+ lines	Quality verification, measurement techniques
D: PowerShell Integration	5 automation scripts	1,100+ lines	Building automation, batch processing

Learning Paths

Path 1: Complete Mastery (18-22 hours)

-> Lessons 1-11 + All Appendices

Best for: Complete skill development, comprehensive understanding

Path 2: Design Focus (12-15 hours)

-> Lessons 1-3, 6-8, 11 + Appendices A, B, C

Best for: Experienced makers new to programmatic CAD

Path 3: Project-Based (14-18 hours)

-> Lessons 1-5 (Foundations) -> 6 (Keycap) -> 7 (Stand) -> 8 (Bins) -> 9 (Automation) -> 10 (Troubleshooting) -> 11 (Leadership)

Best for: Learning through building

Path 4: Safety & Printing (10-12 hours)

-> Lessons 1, 2, 5, 6, 10 + Appendices A, B, C

Best for: Focus on practical printing and quality

3dm Command Reference

Essential Commands

```
# Setup
./3dm setup # Initial configuration

# Development
3dm edit-model file.scad # Open in editor
3dm build src/main.scad # Generate STL from SCAD

# Inspection
3dm describe file.scad # Text analysis (AI if configured)
3dm preview file.scad # Generate 2D tactile preview

# Optimization
3dm orient file.scad # Suggest print orientation

# Production
3dm slice file.scad # Generate G-code
3dm send build/main.gcode # Send to printer

# Libraries
3dm lib list # Show available libraries
3dm lib install BOSL2 # Install a library
```

Command Chaining

```
# Sequential with error handling
3dm build src/main.scad && 3dm slice src/main.scad && echo "Ready to print"

# Loop through files
for f in src/*.scad; do 3dm build "$f" && 3dm slice "$f"; done
```

OpenSCAD Quick Reference

Primitives

```
cube([width, height, depth], center=false);
sphere(r=radius, $fn=32);
cylinder(r=radius, h=height, $fn=32);
```

Transforms

```
translate([x, y, z]) { ... }
rotate([x_deg, y_deg, z_deg]) { ... }
scale([x, y, z]) { ... }
```

Boolean Operations

```
union() { shape1; shape2; } // Combine
difference() { shape1; shape2; } // Subtract
intersection() { shape1; shape2; } // Keep overlap
```

Modules

```
module my_shape(size) {
  cube([size, size, size]);
}
my_shape(20); // Call module
```

Parameters

```
width = 50; // mm
height = 30; // mm
inner = width - 2*wall;
```

Projects Reference

Project 1: Parametric Keycap (Lesson 6)

Key Parameters:

```
key_size = 18; // mm
key_height = 12; // mm
wall = 1.2; // mm
letter = "A"; // Character
```

Variants to Try:

- Small: 12mm, 10mm height
- Medium: 18mm, 12mm height
- Large: 24mm, 14mm height

Files:

- Code: Lesson 6 (Keycap section)
- Output: keycap_X.scad, keycap_X.stl

Project 2: Phone Stand (Lesson 7)

Key Parameters:

```
phone_width = 75; // mm
base_width = 85; // mm
angle = 60; // degrees
lip_height = 15; // mm
```

Configurations:

Phone	Width	Angle	Result
iPhone	60mm	60	Portrait viewing
iPad	100mm	40	Landscape viewing
Tablet	150mm	35	Document viewing

Files:

- Code: Lesson 7 (Phone Stand section)
- Output: stand_X.stl, stand_X.gcode

Project 3: Stackable Bins (Lesson 8)

Key Parameters:

```
bin_w = 80; // width (mm)
bin_d = 120; // depth (mm)
bin_h = 60; // height (mm)
wall = 2; // thickness (mm)
stack_clear = 0.6; // tolerance (mm) - CRITICAL
```

Tolerance Testing:

```
stack_clear = 0.4mm -> Too tight (hard to stack)
stack_clear = 0.6mm -> Ideal (smooth fit)
stack_clear = 0.8mm -> Too loose (unstable)
```

Files:

- Code: Lesson 8 (Bins section)
- Output: bin_*.stl, tolerance_matrix.md

Code Template Library

Generic Parametric Part Template

```
// ====== PARAMETERS (customize here) ======
param1 = 50; // mm
param2 = 30; // mm
param3 = 5; // mm
$fn = 32; // Resolution (lower = faster)
// ====== CALCULATED PARAMETERS ======
derived_param = param1 - 2*param3;
// ====== MODULES ======
module my_part() {
  cube([param1, param2, param3]);
}
// ====== MAIN ======
my_part();
```

Hollow Box Template

```
outer_size = 50;
inner_size = 40;
wall = 5;
difference() {
  cube([outer_size, outer_size, outer_size]);
  translate([wall, wall, wall])
    cube([inner_size, inner_size, inner_size]);
}
```

Batch Build Script Template

```
#!/bin/bash
# batch_build.sh

for scad in src/*.scad; do
  name=$(basename "$scad" .scad)
  echo "Building: $name"
  3dm build "$scad" || continue
  cp "build/main.stl" "build/${name}.stl"
done
```

Troubleshooting Quick Fixes

Problem: Model won't build

Diagnosis:

```
3dm describe file.scad
# Look for error messages
```

Common Fixes:

- Check syntax (missing semicolons, parentheses)
- Look for non-manifold geometry
- Use `$fn=12` for faster test renders

Problem: Parts don't fit together

Diagnosis:

- Print and test fit
- Measure with calipers

Solution:

- Adjust `stack_clear` (smaller = tighter)
- Increase `wall` thickness
- Test with tolerance matrix

Problem: Embossed text looks bad

Diagnosis:

- Check preview in slicer
- Use `3dm preview` for tactile version

Solution:

- Increase `letter_raise` (deeper emboss)
- Use larger `$fn` in `text()`
- Simplify character or use different size

Problem: Print fails

Diagnosis:

- Check slicer layer preview
- Verify bed adhesion and temperature

Solution:

- Check pre-print checklist (Lesson 5)
- Adjust print temperature
- Verify bed is level and clean

Assessment Checklist

Lesson Completion Criteria

- ☐ Watched/read entire lesson
- ☐ Completed all step-by-step tasks
- ☐ Reached all checkpoints
- ☐ Answered all quiz questions (self-assessed)
- ☐ Attempted at least 3 extension problems
- ☐ Documented findings

Project Completion Criteria

- ☐ Code builds without errors
- ☐ All parameters functional

☐ STL generated and inspected
☐ Measurements documented
☐ Assembly tested (if multi-part)
☐ README or documentation included

Quality Standards

☐ Code is well-commented
☐ Parameters have clear names and units
☐ Modules are reusable
☐ Design follows DRY principle
☐ Documentation is complete

Resources & Links

Official Docs

- OpenSCAD Manual
- 3DMake GitHub
- BOSL2 Library

Tutorials

- OpenSCAD Basics
- 3D Printing Guide

Community

- OpenSCAD Forums
- r/3Dprinting

Tips & Tricks

Debugging

- Lower `$fn` to 8-12 for faster renders during development
- Use `3dm describe` frequently to catch issues early
- Test components individually before assembling
- Generate `3dm preview` for 2D tactile verification

Design

- Keep parameters at the top of file for easy modification
- Use descriptive names (not `w`, use `width`)
- Include units in comments
- Document parameter ranges (e.g., `// 0-100 mm`)

Organization

- Use `src/` for SCAD files, `lib/` for modules, `build/` for outputs
- Create variants by copying files and renaming
- Use bash scripts for batch operations
- Archive successful builds with timestamps

Accessibility

- Always use `3dm describe` to verify non-visual usability
- Generate `3dm preview` for tactile inspection
- Document measurements clearly
- Test assembly without visual guidance

Glossary

Term	Definition
CSG	Constructive Solid Geometry - combining shapes using union/difference
Manifold	Water-tight geometry with clear inside/outside
Parametric	Driven by variables; changing parameters updates design
Tolerance	Acceptable variation in dimensions
Stack-up	Cumulative error from multiple tolerances
Module	Reusable code block in OpenSCAD
$fn	Resolution parameter (higher = more detail but slower)
G-code	Machine instructions for 3D printer
STL	3D model file format for printing

Quick Answers

Q: Where do I put my SCAD files?
A: In the `src/` folder of your 3dMake project

Q: How do I test if my design will fit?
A: Use the tolerance testing matrix; print variants with different parameters

Q: What should I measure after printing?
A: Critical dimensions and compare to design specifications

Q: How do I fix non-manifold geometry?
A: Use the 0.001 offset rule: `translate([0, 0, 0.001])` before subtracting

Q: Can I combine multiple SCAD files?
A: Yes, use `include <path/to/file.scad>` or `use <path/to/file.scad>`

Q: How do I make designs accessible?
A: Use `3dm describe` and `3dm preview` and include written measurements/descriptions

BackMatter

This section contains supplementary material and appendices that support the main book.

- **Command Line Appendices** — guidance and integration notes for using the command line with SCAD workflows.
  - Appendix A: CMD Integration for SCAD Workflows
  - Appendix B: PowerShell Integration for SCAD Workflows
  - Appendix C: PowerShell Integration for SCAD Workflows
- **3D Make and OpenSCAD Appendices** — deeper references for slicing, materials, tolerances and advanced OpenSCAD topics.
  - Appendix A: Comprehensive Slicing Guide - All Major Slicers
  - Appendix B: Material Properties & Selection Guide
  - Appendix C: Tolerance Testing & Quality Assurance Matrix
  - Appendix D: Advanced OpenSCAD Concepts
- **Further Reading** — curated list of books, articles, tools, and resources to continue learning.
  - Further Reading
- **Glossary** — a concise list of terms and definitions used throughout the book.
  - Glossary
- **Index** — an alphabetical index of topics for quick lookup.
  - Index

How to use these appendices

- Each appendix is intended as a focused reference you can consult when you need extra detail beyond the course material.

- They are ordered in the book's SUMMARY so they will appear after the main lessons and can be built separately or merged into the full PDF output.
- **Glossary and Index:**
  - The glossary collects short definitions and cross-references to chapters where terms appear.
  - The index is produced as a separate backmatter component and may be generated during PDF build using pandoc's raw LaTeX index commands or a separate index generation step (for example, tagging terms with index markers and running `makeindex` during the LaTeX build).

About the Author

See the author's notes and contributor information: About the Author

Appendices

Comprehensive reference materials, guides, and supplemental resources for the OpenSCAD and 3dMake curriculum.

Command Line Interface

- Appendix A: Comprehensive Slicing Guide - All Major Slicers - Complete reference for PrusaSlicer, Bambu Studio, Cura, and OrcaSlicer configuration
- Appendix B: Material Properties & Selection Guide - Detailed material reference including shrinkage data, print settings, and properties
- Appendix C: Tolerance Testing & Quality Assurance Matrix - Comprehensive QA procedures and tolerance validation methods
- Appendix D: PowerShell Integration for SCAD Workflows - Batch processing, automation scripts, and advanced workflow integration

Appendix A: Command Line (CMD/Batch) Integration for SCAD Workflows

This appendix shows how traditional command-line (Windows CMD / batch) scripting automates SCAD workflows without requiring PowerShell. It mirrors the patterns in Appendix D but provides examples and idioms for `cmd` / batch files and systems that prefer native Windows command prompt scripts.

Overview: Why Automate with CMD/Batch?

- Minimal dependencies: works on basic Windows installations.
- Easy to call from other tools and CI systems that expect `.bat` helpers.
- Useful for environments where PowerShell policy restrictions exist.

Prerequisites & Setup

Required Software

```
where openscad :: OpenSCAD (path in PATH or full path required)
where prusa-slicer :: PrusaSlicer (or your slicer)
where python :: Python (optional)
```

Directory Structure (Windows style)

```
C:\Projects\3dMake\
+------ src\
+------ stl\
+------ gcode\
+------ logs\
+------ scripts\
      +------ build.bat
      +------ batch_build.bat
```

Notes on CMD vs PowerShell

- CMD/batch has more limited error handling and no structured objects; rely on return codes and file checks.
- Use full executable paths to avoid PATH surprises.

Basic Workflow: Single-file build (batch)

Create `build.bat` (simple example):

```
@echo off
rem build.bat - Convert SCAD -> STL -> G-code (minimal)
setlocal enabledelayedexpansion
set PROJECTROOT=%~dp0\..\
set SCADFILE=%1
if "%SCADFILE%"=="" (
  echo Usage: build.bat file.scad
  exit /b 1
)
set SCADPATH=%PROJECTROOT%src\%SCADFILE%
set STLPATH=%PROJECTROOT%stl\%~n1.stl
set GCODEPATH=%PROJECTROOT%gcode\%~n1.gcode

rem Export STL using OpenSCAD
"C:\Program Files\OpenSCAD\openscad.exe" -o "%STLPATH%" "%SCADPATH%"
if not exist "%STLPATH%" (
  echo ERROR: STL not created
  exit /b 1
)

rem Slice (example) - adjust path to slicer
"C:\Program Files\Prusa3D\PrusaSlicer\prusa-slicer.exe" --load-config-file " →
    ↪ %~dp0\default.ini" -o "%GCODEPATH%" "%STLPATH%"
if not exist "%GCODEPATH%" (
  echo ERROR: G-code not created
  exit /b 1
)

echo BUILD COMPLETE: %GCODEPATH%
endlocal
```

Batch (directory) build

`batch_build.bat` can iterate files and call `build.bat`:

```
@echo off
set PROJECTROOT=%~dp0\..\
for /r "%PROJECTROOT%src" %%f in (*.scad) do (
  echo Processing %%~nxf
  call "%~dp0build.bat" "%%~nxf"
)
```

Parametric Sweeps (simple templating)

Because batch is limited for text templating, a common pattern is to use small helper scripts with `sed`/`python` or write a minimal templating helper in a language such as Python. Example invocation in batch:

```
rem Example: call a Python script to generate variants, then build
python "%~dp0/generate_variants.py" "%PROJECTROOT%src\bracelet_holder.scad"
call "%~dp0/batch_build.bat"
```

Logging and CSV output

Batch scripts can append simple CSV lines using `echo` redirection:

```
echo %DATE% %TIME%,%~n1,Success >> "%PROJECTROOT%logs\build_history.csv"
```

Send to Printer (USB copy)

```
rem Copy gcode to removable drive (E: example)
copy "%GCODEPATH%" E:\
```

Monitoring (networked printer) - using curl or PowerShell helper

`curl` on Windows or a small PowerShell one-liner may be used to query APIs. For pure CMD, ship a small helper .exe (curl) or call `powershell -Command "Invoke-RestMethod ..."`.

Best Practices

- Use full paths for all tools.
- Check return codes (`if errorlevel 1`) after each step.
- Keep batch scripts small; delegate complex text processing to Python/PowerShell.

- Log to simple CSVs for human and machine parsing.

Appendix B: PowerShell Integration for SCAD Workflows

This appendix shows how PowerShell (command-line scripting) streamlines 3D design workflows, automating the repetitive tasks from design through printing. It bridges PowerShell_Foundation concepts with 3dMake_Foundation practical applications.

Referenced in: Lessons 9 (Automation), 10 (Mastery), and any lesson requiring batch operations

Prerequisites: Completion of PowerShell_Foundation (Lessons 1-6)

Overview: Why Automate SCAD Workflows?

Manual Workflow (Time-Consuming)

1. Open SCAD manually -> Edit parameters -> Save -> Export STL
2. Open Slicer manually -> Load STL -> Adjust settings -> Slice -> Export G-code
3. Transfer G-code to printer via USB
4. Record results in notebook
5. Repeat for next variation (variant 2, variant 3, etc.)

Time: ~20-30 minutes per design iteration

Automated Workflow (Fast & Reproducible)

1. Create PowerShell script with design parameters
2. Script automatically:
    - Generates SCAD code
    - Exports to STL
    - Slices to G-code
    - Transfers to printer
    - Logs results
3. Run script once, walk away
4. Check results later

Time: 2-3 minutes per iteration (plus print time)

Benefits of Automation

- [YES] Speed: 10x faster for batch operations
- [YES] Consistency: No manual errors
- [YES] Reproducibility: Same settings every time
- [YES] Scalability: Test 5, 10, or 100 variations easily

- [YES] Logging: Automatic documentation
- [YES] Accessibility: Screen reader captures script output (not UI clicks)

Prerequisites & Setup

Required Software

```
# Check what you have installed
where openscad # OpenSCAD
where prusa-slicer # PrusaSlicer (or your slicer)
where python3 # Python (optional, for advanced scripts)
```

Directory Structure Create a project directory:

```
C:\Projects\3dMake\
+------ src/
| +------ bracelet_holder.scad
| +------ phone_stand.scad
| +------ stackable_bins.scad
+------ stl/
| +------ bracelet_holder.stl
| +------ phone_stand.stl
| +------ stackable_bins.stl
+------ gcode/
| +------ bracelet_holder.gcode
| +------ phone_stand.gcode
| +------ stackable_bins.gcode
+------ logs/
| +------ batch_2024-01-15.log
| +------ print_history.csv
+------ scripts/
    +------ build.ps1 (main build script)
    +------ batch_test.ps1 (test variations)
    +------ monitor.ps1 (monitor printer)
```

PowerShell Execution Policy

```
# Check current policy
Get-ExecutionPolicy
# If it's "Restricted", change it (requires admin)
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
# Verify change
Get-ExecutionPolicy
```

Basic Workflow Automation

Script 1: Single-File Build This script takes a SCAD file and converts it through the entire workflow:

```
# build.ps1 - Convert SCAD -> STL -> G-code
param(
    [string]$ScadFile, # Input: bracelet_holder.scad
    [string]$OutputDir = ".\", # Output directory
    [string]$SlicerConfig = "default"
)
# Set up paths
$ProjectRoot = "C:\Projects\3dMake"
$ScadFile = Join-Path $ProjectRoot "src" $ScadFile
$StlFile = Join-Path $ProjectRoot "stl" ([System.IO.Path]:: →
    ↪ GetFileNameWithoutExtension($ScadFile) + ".stl")
$GcodeFile = Join-Path $ProjectRoot "gcode" ([System.IO.Path]:: →
    ↪ GetFileNameWithoutExtension($ScadFile) + ".gcode")
```

```
$LogFile = Join-Path $ProjectRoot "logs" "build_$(Get-Date -Format 'yyyy-MM-
    dd').log"
# Log function
function Write-Log {
    param([string]$Message)
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $logEntry = "[$timestamp] $Message"
    Write-Host $logEntry
    Add-Content -Path $LogFile -Value $logEntry
}
Write-Log "Starting build for $($ScadFile | Split-Path -Leaf)"
# Step 1: Export STL
Write-Log "Step 1: Exporting SCAD to STL..."
$startTime = Get-Date
& "C:\Program Files\OpenSCAD\openscad.exe" `
    -o "$StlFile" `
    "$ScadFile"
$duration = (Get-Date) - $startTime
Write-Log " STL exported in $($duration.TotalSeconds) seconds: $StlFile"
# Verify STL exists
if (-not (Test-Path $StlFile)) {
    Write-Log "[NO] ERROR: STL file not created"
    exit 1
}
# Step 2: Slice to G-code
Write-Log "Step 2: Slicing STL to G-code..."
$startTime = Get-Date
& "C:\Program Files\Prusa3D\PrusaSlicer\prusa-slicer.exe" `
    --load-config-file "$SlicerConfig" `
    --export-gcode "$GcodeFile" `
    "$StlFile"
$duration = (Get-Date) - $startTime
Write-Log " G-code generated in $($duration.TotalSeconds) seconds:
    $GcodeFile"
# Verify G-code exists
if (-not (Test-Path $GcodeFile)) {
    Write-Log "[NO] ERROR: G-code file not created"
    exit 1
}
# Step 3: Get file info
$stlSize = (Get-Item $StlFile).Length / 1MB
$gcodeSize = (Get-Item $GcodeFile).Length / 1MB
Write-Log "File sizes: STL=$([math]::Round($stlSize, 2))MB, G-code=$([math
    ]::Round($gcodeSize, 2))MB"
Write-Log " BUILD COMPLETE"
Write-Log "Ready to print: $GcodeFile"
```

Usage:

```
# Run the script
.\build.ps1 -ScadFile "bracelet_holder.scad"
# Output example:
# [2024-01-15 14:30:22] Starting build for bracelet_holder.scad
# [2024-01-15 14:30:23] Step 1: Exporting SCAD to STL...
# [2024-01-15 14:30:25] STL exported in 2.41 seconds: ...
# [2024-01-15 14:30:26] Step 2: Slicing STL to G-code...
# [2024-01-15 14:30:28] G-code generated in 2.31 seconds: ...
# [2024-01-15 14:30:28] BUILD COMPLETE
```

Script 2: Batch Build (Multiple Files)

```
# batch_build.ps1 - Build all SCAD files in a directory
param(
    [string]$SourceDir = "C:\Projects\3dMake\src",
    [string]$SlicerConfig = "default"
)
$ProjectRoot = Split-Path $SourceDir
$buildScript = Join-Path $ProjectRoot "scripts" "build.ps1"
# Find all SCAD files
```

```
$scadFiles = Get-ChildItem -Path $SourceDir -Filter "*.scad" -Recurse
Write-Host "Found $($scadFiles.Count) SCAD files"
if ($scadFiles.Count -eq 0) {
    Write-Host "No SCAD files found in $SourceDir"
    exit 1
}
# Process each file
$results = @()
foreach ($file in $scadFiles) {
    Write-Host "`n--- Processing: $($file.Name) ---"
    $startTime = Get-Date
    # Run build script for this file
    & $buildScript -ScadFile $file.Name -SlicerConfig $SlicerConfig
    $duration = (Get-Date) - $startTime
    $results += [PSCustomObject]@{
        FileName = $file.Name
        DurationSeconds = $duration.TotalSeconds
        Status = "Success"
        Timestamp = Get-Date
    }
}
# Summary report
Write-Host "`n=== BATCH BUILD SUMMARY ==="
Write-Host "Files processed: $($results.Count)"
Write-Host "Total time: $([math]::Round(($results | Measure-Object -Property →
    ↪  DurationSeconds -Sum).Sum, 1)) seconds"
# Save results to CSV
$csvPath = Join-Path $ProjectRoot "logs" "batch_$(Get-Date -Format 'yyyy-MM- →
    ↪ dd-HHmmss').csv"
$results | Export-Csv -Path $csvPath -NoTypeInformation
Write-Host "Results saved: $csvPath"
```

Usage:

```
.\batch_build.ps1 -SourceDir "C:\Projects\3dMake\src"
# Processes all .scad files automatically
```

Parametric Design Variation Testing

The Problem: Testing Multiple Parameter Values You want to test the same design with different parameters (e.g., peg diameter: 5mm, 6mm, 7mm):

Manually:

1. Open bracelet_holder.scad
2. Change peg_diameter = 5
3. Save, export, slice, print
4. Change peg_diameter = 6
5. Save, export, slice, print
6. Change peg_diameter = 7
7. Save, export, slice, print Time: 30 minutes

Automated Solution: Parametric Sweep

```
# parametric_sweep.ps1 - Test multiple parameter values
param(
    [string]$TemplateScad = "C:\Projects\3dMake\src\bracelet_holder.scad",
    [hashtable]$ParameterRanges = @{
        "peg_diameter" = @(5, 5.5, 6, 6.5, 7)
        "holder_width" = @(120, 127, 135)
    }
)
```

```
$ProjectRoot = "C:\Projects\3dMake"
$variantsDir = Join-Path $ProjectRoot "variants"
$buildScript = Join-Path $ProjectRoot "scripts" "build.ps1"
$logFile = Join-Path $ProjectRoot "logs" "parametric_sweep_$(Get-Date - →
    ↪ Format 'yyyy-MM-dd-HHmmss').csv"
# Create variants directory
New-Item -ItemType Directory -Path $variantsDir -Force | Out-Null
# Read template
$template = Get-Content $TemplateScad -Raw
# Generate variants
$results = @()
$variantNum = 0
# For each diameter value
foreach ($dia in $ParameterRanges["peg_diameter"]) {
    # For each width value
    foreach ($width in $ParameterRanges["holder_width"]) {
        $variantNum++
        $variantName = "variant_dia${dia}_width${width}"
        $variantScad = Join-Path $variantsDir "$variantName.scad"
        # Create variant SCAD file with parameters
        $content = $template `
            -replace 'peg_diameter = [\d.]+', "peg_diameter = $dia" `
            -replace 'holder_width = [\d.]+', "holder_width = $width"
        $content | Set-Content $variantScad
        Write-Host "Generated: $variantName"
        # Build variant (STL + G-code)
        $startTime = Get-Date
        # Export to STL
        & "C:\Program Files\OpenSCAD\openscad.exe" `
            -o (Join-Path $ProjectRoot "stl" "$variantName.stl") `
            $variantScad
        $duration = (Get-Date) - $startTime
        # Record result
        $results += [PSCustomObject]@{
            Variant = $variantName
            PegDiameter = $dia
            HolderWidth = $width
            BuildTimeSeconds = $duration.TotalSeconds
            Status = "Success"
            Timestamp = Get-Date
        }
        Write-Host " Built in $($duration.TotalSeconds) seconds"
    }
}
# Save results
$results | Export-Csv -Path $logFile -NoTypeInformation
Write-Host "`nParametric sweep complete: $($results.Count) variants  →
    ↪ generated"
Write-Host "Results: $logFile"
# Summary statistics
Write-Host "`n=== PARAMETER RANGES TESTED ==="
Write-Host "Peg diameters: $($ParameterRanges['peg_diameter'] -join ', ')"
Write-Host "Holder widths: $($ParameterRanges['holder_width'] -join ', ')"
Write-Host "Total combinations: $($results.Count)"
```

Usage:

```
$ranges = @{
    "peg_diameter" = @(5, 5.5, 6, 6.5, 7)
    "holder_width" = @(120, 127, 135)
}
.\parametric_sweep.ps1 -TemplateScad "bracelet_holder.scad" -ParameterRanges →
    ↪ $ranges
# Generates 15 variants automatically (5 x 3)
```

Output CSV:

```
Variant,PegDiameter,HolderWidth,BuildTimeSeconds,Status,Timestamp
variant_dia5_width120,5,120,2.4,Success,2024-01-15 14:35:22
```

```
variant_dia5_width127,5,127,2.5,Success,2024-01-15 14:35:25
variant_dia5_width135,5,135,2.3,Success,2024-01-15 14:35:28
...
```

Automated Print Documentation

Script: Print Logging & Quality Tracking

```
# log_print.ps1 - Document each print with metadata
param(
    [string]$GcodeFile,
    [string]$ProjectName,
    [string]$Material = "PLA",
    [string]$Notes = ""
)
$ProjectRoot = "C:\Projects\3dMake"
$printLogCsv = Join-Path $ProjectRoot "logs" "print_history.csv"
# Get G-code file info
$gcodeInfo = Get-Item $GcodeFile
$gcodeSize = $gcodeInfo.Length / 1MB
$estimatedTime = EstimateTime $GcodeFile # See function below
# Calculate estimated cost
$weightG = EstimateWeight $GcodeFile
$materialCostPerKg = 20 # PLA at $20/kg
$estimatedCost = ($weightG / 1000) * $materialCostPerKg
# Create log entry
$entry = [PSCustomObject]@{
    PrintID = "PRINT_$(Get-Date -Format 'yyyyMMdd_HHmmss')"
    ProjectName = $ProjectName
    Date = Get-Date
    Material = $Material
    GcodeFile = $gcodeFile
    GcodeSizeMB = [math]::Round($gcodeSize, 2)
    EstimatedTimeHours = [math]::Round($estimatedTime / 3600, 2)
    EstimatedWeightG = [math]::Round($weightG, 1)
    EstimatedCostUSD = [math]::Round($estimatedCost, 2)
    Notes = $Notes
}
# Append to CSV
if (Test-Path $printLogCsv) {
    $entry | Export-Csv -Path $printLogCsv -NoTypeInformation -Append
} else {
    $entry | Export-Csv -Path $printLogCsv -NoTypeInformation
}
Write-Host "Print logged:"
Write-Host " Project: $ProjectName"
Write-Host " File: $($gcodeInfo.Name)"
Write-Host " Estimated time: $([math]::Round($estimatedTime / 3600, 2)) →
    ↪ hours"
Write-Host " Estimated weight: $([math]::Round($weightG, 1))g"
Write-Host " Estimated cost: $$([math]::Round($estimatedCost, 2))"
# Function to estimate print time from G-code (simplified)
function EstimateTime {
    param([string]$GcodeFile)
    $lines = @(Get-Content $GcodeFile | Select-String "^G1" | Measure-Object) →
        ↪ .Count
    # Rough estimate: 10 lines per second
    return $lines / 10
}
# Function to estimate weight from G-code (simplified)
function EstimateWeight {
    param([string]$GcodeFile)
    # G-code filament estimate (if supported by your slicer)
    $content = Get-Content $GcodeFile -Raw
    if ($content -match "filament used = ([\d.]+)") {
        return [double]$matches[1]
    } else {
        return 0 # Fallback
    }
}
```

Usage:

```
.\log_print.ps1 `
   -GcodeFile "C:\Projects\3dMake\gcode\bracelet_holder.gcode" `
   -ProjectName "Bracelet Holder" `
   -Material "PLA" `
   -Notes "Final design, tested with actual bracelets"
```

Printer Communication & Monitoring

Script: Send G-code to Printer (USB)

```
# send_to_printer.ps1 - Copy G-code to printer USB
param(
    [string]$GcodeFile,
    [string]$PrinterUSBLetter = "E" # E:, F:, G:, etc.
)
$printerDrive = "${PrinterUSBLetter}:\"
# Check if USB drive connected
if (-not (Test-Path $printerDrive)) {
    Write-Host "[NO] ERROR: Printer USB not found at $PrinterUSBLetter"
    Write-Host "Available drives:"
    Get-PSDrive -PSProvider FileSystem | Where-Object Name -Like "[D-Z]" |  →
         ↪ Select-Object Name
    exit 1
}
# Copy file
$filename = Split-Path $GcodeFile -Leaf
$destination = Join-Path $printerDrive $filename
Copy-Item -Path $GcodeFile -Destination $destination -Force
Write-Host " G-code copied to printer USB:"
Write-Host " From: $GcodeFile"
Write-Host " To: $destination"
Write-Host "`nNext steps:"
Write-Host " 1. Eject USB safely from computer"
Write-Host " 2. Insert USB into printer"
Write-Host " 3. Select file on printer screen"
Write-Host " 4. Press print"
```

Usage:

```
.\send_to_printer.ps1 -GcodeFile "C:\Projects\3dMake\gcode\bracelet_holder. →
    ↪ gcode" -PrinterUSBLetter "E"
```

Script: Monitor Printer Status (Network Printers)

```
# monitor_printer.ps1 - Check printer status via API
param(
    [string]$PrinterIP,
    [int]$CheckInterval = 30, # seconds
    [int]$MaxChecks = 240 # 2 hours max
)
$printerUrl = "[https://example.com](https://example.com)
$checksPerformed = 0
Write-Host "Monitoring printer at $PrinterIP"
Write-Host "Check interval: $CheckInterval seconds"
Write-Host "Ctrl+C to stop`n"
while ($checksPerformed -lt $MaxChecks) {
    try {
        # Get printer status
        $response = Invoke-WebRequest -Uri $printerUrl -UseBasicParsing - →
             ↪ ErrorAction Stop
        $status = $response.Content | ConvertFrom-Json
        $state = $status.state.text
        $progress = $status.progress.completion
        $timeRemaining = $status.progress.printTimeLeft
        Write-Host "[$((Get-Date).ToString("HH:mm:ss"))] State: $state | Progress: $progress% | Time remaining: $timeRemaining seconds"
        # If print completed, exit loop
        if ($state -eq "Operational") {
            Write-Host "`n Print complete!"
            break
        }
    } catch {
        Write-Host "Connection error: $_"
    }
    Start-Sleep -Seconds $CheckInterval
    $checksPerformed++
}
if ($checksPerformed -ge $MaxChecks) {
    Write-Host "Max monitoring time reached. Stopping."
}
```

Usage:

```
.\monitor_printer.ps1 -PrinterIP "192.168.1.100" -CheckInterval 30
```

Complete Workflow Integration

Master Script: Design -> Print -> Log

```
# full_workflow.ps1 - Complete automation from design to printing
param(
    [string]$ProjectName,
    [string]$ScadFile,
    [string]$Material = "PLA",
    [string]$Notes = "",
    [switch]$SendToPrinter,
    [string]$PrinterUSBLetter = "E"
)
$ProjectRoot = "C:\Projects\3dMake"
Write-Host "=== 3dMake Full Workflow Automation ==="
Write-Host "Project: $ProjectName"
Write-Host ""
# Step 1: Build (SCAD -> STL -> G-code)
Write-Host "Step 1: Building..."
& "$ProjectRoot\scripts\build.ps1" -ScadFile $ScadFile
if ($LASTEXITCODE -ne 0) { exit 1 }
# Step 2: Log the print
Write-Host "`nStep 2: Logging print metadata..."
$gcodeFile = Join-Path $ProjectRoot "gcode" ([System.IO.Path]::GetFileNameWithoutExtension($ScadFile) + ".gcode")
& "$ProjectRoot\scripts\log_print.ps1" `
    -GcodeFile $gcodeFile `
    -ProjectName $ProjectName `
    -Material $Material `
    -Notes $Notes
# Step 3: Send to printer (optional)
if ($SendToPrinter) {
    Write-Host "`nStep 3: Sending to printer..."
    & "$ProjectRoot\scripts\send_to_printer.ps1" `
        -GcodeFile $gcodeFile `
        -PrinterUSBLetter $PrinterUSBLetter
}
Write-Host "`n Workflow complete!"
```

Usage:

```
# Full workflow with automatic printer transfer
.\full_workflow.ps1 `
    -ProjectName "Bracelet Holder" `
    -ScadFile "bracelet_holder.scad" `
    -Material "PLA" `
    -Notes "Final design, v2 with improved peg strength" `
    -SendToPrinter `
    -PrinterUSBLetter "E"
```

PowerShell Skills Applied to SCAD

Lesson Mapping: PowerShell -> SCAD Workflows

PowerShell Lesson	SCAD Application	Example
PS 1: Navigation	Working with project directories	Using `cd` to navigate src/ -> stl/ -> gcode/
PS 2: File Manipulation	Copying, organizing SCAD files	Copy design files, organize variants by date
PS 3: Piping & Objects	Pass data between scripts	`$results	Export-Csv` exports analysis
PS 4: Variables & Aliases	Parameterize SCAD workflows	Variables store file paths, material types, etc.
PS 5: Functions & Modules	Reusable automation blocks	Build, slice, log = functions in one script
PS Unit Test	Verify workflow correctness	Test that STL file exists before slicing

Example: File Navigation with SCAD Projects

```
# Navigate between SCAD project folders
$ProjectRoot = "C:\Projects\3dMake"
# Go to source directory
cd "$ProjectRoot\src"
Get-ChildItem -Filter "*.scad" # List all SCAD files
# Process each file
foreach ($file in Get-ChildItem -Filter "*.scad") {
    Write-Host "Processing: $($file.Name)"
    # Do something with each file
}
# Go to output directory and check results
cd "$ProjectRoot\stl"
Get-ChildItem -Filter "*.stl" | Measure-Object -Sum
```

Example: Working with Piping & Objects

```
# Build all files, then get statistics
Get-ChildItem -Path "$ProjectRoot\src" -Filter "*.scad" |
    ForEach-Object {
        # Build each one
        & .\build.ps1 -ScadFile $_.Name
    } |
    # Analyze results
    Group-Object -Property Status |
    Select-Object Name, Count
```

Accessibility Considerations

Why PowerShell for SCAD?

1. Screen Reader Friendly: Console output = text (not UI clicks)
2. Scriptable: Can run overnight, results logged
3. Auditable: Every step written to log file
4. Shareable: Scripts document the process for others
5. Testable: Can verify each step independently

Example: Accessible Build Output

```
[2024-01-15 14:30:22] Starting build for bracelet_holder.scad
[2024-01-15 14:30:23] Step 1: Exporting SCAD to STL...
[2024-01-15 14:30:25] STL exported in 2.41 seconds
[2024-01-15 14:30:26] Step 2: Slicing STL to G-code...
[2024-01-15 14:30:28] G-code generated in 2.31 seconds
[2024-01-15 14:30:28] File sizes: STL=2.15MB, G-code=4.32MB
[2024-01-15 14:30:28] BUILD COMPLETE

Ready to print: C:\Projects\3dMake\gcode\bracelet_holder.gcode
```

All information is text-based and sequential-perfectly accessible to screen readers.

Best Practices & Tips

1. Always Log Everything

```
function Write-Log {
    param(
        [string]$Message,
        [string]$LogFile
    )
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $entry = "[$timestamp] $Message"
    Write-Host $entry
    Add-Content -Path $LogFile -Value $entry
}
```

2. Verify Steps Succeed Before Continuing

```
# Don't just run commands-check they worked
& "C:\Program Files\OpenSCAD\openscad.exe" -o "$StlFile" "$ScadFile"
if (-not (Test-Path $StlFile)) {
    Write-Log "ERROR: STL creation failed"
    exit 1
}
Write-Log " STL created successfully"
```

3. Make Scripts Reusable

```
# BAD: Hardcoded paths
$scadFile = "C:\Users\John\Desktop\bracelet_holder.scad"
# GOOD: Parameters with defaults
param(
    [string]$ScadFile = "bracelet_holder.scad",
    [string]$ProjectRoot = "C:\Projects\3dMake"
)
$fullPath = Join-Path $ProjectRoot "src" $ScadFile
```

4. Use Configuration Files

```
# config.ps1 - Centralized settings
$Config = @{
    ProjectRoot = "C:\Projects\3dMake"
    OpenSCADPath = "C:\Program Files\OpenSCAD\openscad.exe"
    SlicerPath = "C:\Program Files\Prusa3D\PrusaSlicer\prusa-slicer.exe"
    PrinterUSB = "E"
    DefaultMaterial = "PLA"
    DefaultInfill = 20
}
# Use in scripts:
# . .\config.ps1
# & $Config.OpenSCADPath ...
```

Troubleshooting Automated Workflows

Problem	Cause	Solution
Scripts won't run	Execution policy	`Set-ExecutionPolicy -ExecutionPolicy RemoteSigned`
Command not found	Tool not in PATH	Specify full path: `& "C:\Program Files\OpenSCAD\openscad.exe"`
Files not created	Wrong working directory	Use absolute paths with `Join-Path`
Script hangs	Waiting for user input	Disable UI input; use `-o` flag for batch operations
CSV data corrupted	Special characters in notes	Properly quote strings in CSV export
No output/logging	Path doesn't exist	Create directory first: `New-Item -Type Directory -Path ... -Force`

Summary: PowerShell + SCAD Integration Benefits

Time Savings

Task	Manual	Automated	Savings
Single design build	5 min	2.5 min	50%
Test 10 parameter variations	50 min	5 min	90%
Batch processing 20 designs	100 min	10 min	90%

Consistency

- [YES] Same process every time (no missed steps)
- [YES] Reproducible results (exact same settings)
- [YES] Comprehensive logging (documentation automatic)

Scalability

- [YES] 1 design or 1,000 designs = same time commitment
- [YES] Run overnight for batch operations
- [YES] Easy to expand with new features

Accessibility

- [YES] All interaction is text-based (screen reader friendly)
- [YES] Results logged in CSV (machine-readable, sortable)
- [YES] Repeatable (can verify steps later)

Next Steps

1. Try the basic build script (Script 1) with one of your SCAD files
2. Add logging (Script 3) to track what you create
3. Test parametric sweeps (Script 2) with design variations
4. Integrate full workflow (Script 5) for complete automation
5. Customize config.ps1 for your specific setup

Remember: Start simple, test each script individually, then combine them into larger workflows. PowerShell is most powerful when built incrementally!

Appendix C: Git Bash / POSIX Shell Integration for SCAD Workflows

This appendix mirrors Appendix D (PowerShell) but provides examples for Git Bash (mingw) or other POSIX-compatible shells on Windows, macOS, and Linux. Use these scripts when you prefer shell utilities (`bash`, `sed`, `awk`, `curl`) and Unix-like tooling.

Overview: Why Git Bash / POSIX Shell?

- Leverage standard Unix tools for text processing and automation.
- Cross-platform: same scripts often work on Linux/macOS and Windows via Git Bash.
- Good fit for containerized CI like GitHub Actions.

Prerequisites & Setup

Required Software

```
command -v openscad # OpenSCAD
command -v prusa-slicer # PrusaSlicer (or use CLI slicer)
command -v python3 # Python (optional)
command -v curl # curl for API calls
```

Directory Structure (POSIX-style)

```
~/projects/3dMake/
|- src/
|- stl/
|- gcode/
|- logs/
`- scripts/
  |- build.sh
  `- batch_build.sh
```

Basic Workflow: Single-file build (bash)

`scripts/build.sh`:

```bash
#!/usr/bin/env bash
set -euo pipefail
ROOT_DIR="$(cd "$(dirname "$0")/.." && pwd)"
SCAD_FILE="$1"
SCAD_PATH="$ROOT_DIR/src/$SCAD_FILE"
STL_PATH="$ROOT_DIR/stl/${SCAD_FILE%.*}.stl"
GCODE_PATH="$ROOT_DIR/gcode/${SCAD_FILE%.*}.gcode"

echo "Exporting SCAD -> STL: $SCAD_FILE"
"/c/Program Files/OpenSCAD/openscad.exe" -o "$STL_PATH" "$SCAD_PATH" || { echo "STL export failed"; exit 1; }

echo "Slicing STL -> G-code"
"/c/Program Files/Prusa3D/PrusaSlicer/prusa-slicer.exe" --load-config-file "$ROOT_DIR/scripts/default.ini" -o "$GCODE_PATH" "$STL_PATH" || { echo "Slicing failed"; exit 1; }

echo "BUILD COMPLETE: $GCODE_PATH"
```

Notes:

- Paths to GUI apps on Windows may need the `/c/Program\ Files/...` form under Git Bash, or call the native `.exe` with full quoted path.
- On Linux/macOS adjust executable paths accordingly.

Batch Build (directory)

`scripts/batch_build.sh`:

```bash
#!/usr/bin/env bash
set -euo pipefail
ROOT_DIR="$(cd "$(dirname "$0")/.." && pwd)"
find "$ROOT_DIR/src" -name "*.scad" -print0 | while IFS= read -r -d '' file; do
  fname="$(basename "$file")"
  echo "Building $fname"
  "$ROOT_DIR/scripts/build.sh" "$fname"
done
```

Parametric Sweeps

Use `sed` or `python` to template parameter changes, then call `build.sh` for each variant. Example (bash + python hybrid):

```bash
python3 scripts/gen_variants.py "$ROOT_DIR/src/bracelet_holder.scad" --out variants/
for v in variants/*.scad; do
  ./scripts/build.sh "$(basename "$v")"
done
```

Logging & CSV

Append simple CSV lines using `printf`:

```
printf "%s,%s,Success\n" "$(date +'%Y-%m-%d %H:%M:%S')" "${SCAD_FILE%.*}" >> →
    ↪ "$ROOT_DIR/logs/build_history.csv"
```

Send to Printer (USB)

```
# Copy to mounted USB drive (example mount point /media/usb)
cp "$GCODE_PATH" "/media/usb/"
```

Monitoring (network printers)

Use `curl` to call a printer API (e.g., OctoPrint):

```
curl -s "[https://example.com](https://example.com) -H "X-Api-Key: $API_KEY" →
    ↪ | jq .
```

Best Practices

- Use `set -euo pipefail` for safer scripts.
- Use `mktemp` or a `variants/` directory for generated files.
- Keep heavy text processing to `python` if logic becomes complex.
- Make scripts executable (`chmod +x scripts/*.sh`).

Appendices

Comprehensive reference materials, guides, and supplemental resources for the OpenSCAD and 3dMake curriculum.

3dMake Foundation Appendices

- Appendix A: Comprehensive Slicing Guide - All Major Slicers - Complete reference for PrusaSlicer, Bambu Studio, Cura, and OrcaSlicer configuration
- Appendix B: Material Properties & Selection Guide - Detailed material reference including shrinkage data, print settings, and properties
- Appendix C: Tolerance Testing & Quality Assurance Matrix - Comprehensive QA procedures and tolerance validation methods
- Appendix D: PowerShell Integration for SCAD Workflows - Batch processing, automation scripts, and advanced workflow integration

Appendix A: Comprehensive Slicing Guide - All Major Slicers

This appendix covers settings, workflows, and troubleshooting for 7 major 3D printer slicers. Each slicer is represented with:

- Recommended settings for beginners
- Screen-reader-accessible settings explanation
- Common troubleshooting
- Accessibility features built-in
- Command-line usage (for PowerShell integration)

Referenced in: Lessons 5 (Safety), 8 (Design), 10 (Verification)

Overview: What is Slicing?

Slicing converts a 3D model into printer instructions:

```
SCAD Design (bracelet_holder.scad)
   v
Export to STL (3D shape file)
   v
Load into Slicer
   v
Apply settings (temperature, speed, supports, etc.)
   v
Generate G-code (printer instructions)
   v
Send to Printer
   v
Physical part
```

Core Slicing Parameters (All Slicers Share These)

Parameter	What It Does	Typical Range	Impact
Nozzle Temp	Filament melting heat	200-250C	Too cold -> weak; too hot -> oozing
Bed Temp	Build plate heat	50-110C	Helps adhesion; prevents warping
Layer Height	Z-axis precision	0.1-0.4mm	Finer = slower, better detail
Print Speed	Movement velocity	30-150 mm/s	Faster = worse quality; slower = stronger
Infill %	Interior density	10-100%	Higher = stronger + heavier
Support	Temporary scaffolding	On/Off	Required for overhangs >45
Bed Adhesion	First layer stickiness	Brim/Raft/Skirt	Prevents parts lifting mid-print

1. PrusaSlicer (Prusa)

Overview

- Developer: Prusa Research (Czech company)
- Platforms: Windows, Mac, Linux (open-source)

- Best For: Beginner-friendly, excellent support, strong community
- Download: https://www.prusa3d.com/page/prusaslicer_410/
- Accessibility: Good font sizes, text-based profiles

Quick Setup for Beginners Step 1: Install & Select Your Printer

```
1. Open PrusaSlicer
2. Go to "Help" -> "Check for Updates"
3. When prompted, select your printer model
4. Choose default profile (matches printer exactly)
```

Step 2: Load Your STL

```
1. Click "File" -> "Open STL Model"
2. Select your bracelet_holder.stl
3. Model appears in 3D view
```

Step 3: Essential Settings

These are the most important adjustments:

Setting	Location	Beginner Value	Why
Layer Height	Print Settings	0.15mm	Balance speed & quality
Infill	Print Settings	20%	Enough strength; fast print
Support	Print Settings	Yes (if needed)	For overhangs >45
Nozzle Temp	Filament Settings	210C	Default for PLA
Bed Temp	Filament Settings	60C	Standard PLA adhesion
Print Speed	Print Settings	150 mm/s	Balanced quality

Step 4: Preview & Export

```
1. Click "Slice now" (or G-code icon)
2. Left panel shows preview of each layer
3. Look for issues:
   - Supports covering entire model? (OK)
   - Model floating in air? (Not OK-likely error)
   - Top surface quality acceptable?
4. If satisfied, click "Export G-code"
5. Save to USB or send to printer
```

Accessible Parameter Explanations When adjusting settings, use these descriptions to understand what each does:

Layer Height (0.1-0.4mm)

- Lower (0.1mm): Smoother surface, more layers, slower (best for detail)
- Higher (0.4mm): Faster, rougher surface (best for speed)

- Recommendation: 0.15mm for balanced quality

Infill Percentage (10-100%)

- 10%: Fast, uses less filament, weaker (good for prototypes)
- 20%: Good balance (recommended for most prints)
- 50%: Stronger, slower, heavier
- 100%: Solid interior, strongest, slowest (waste of plastic)

Support Type

- None: Fast, good surface finish, but risky if overhangs exist
- Linear (Default): Good balance-easy to remove, provides support
- Grid: Extra strong support, takes longer to remove

First Layer

- Brim: Adds border to help adhesion (recommended for beginners)
- Raft: Sacrificial platform (good if bed isn't level)
- Skirt: Just outline, doesn't help adhesion (fastest)

Command-Line Usage (PowerShell Integration)

```
# Slice a model automatically with PrusaSlicer
$model = "C:\Models\bracelet_holder.stl"
$output = "C:\GCode\bracelet_holder.gcode"
$config = "default" # Use default printer profile
# Run PrusaSlicer in batch mode
& "C:\Program Files\Prusa3D\PrusaSlicer\prusa-slicer.exe" `
    --load-config-file "$config" `
    --export-gcode "$output" `
    "$model"
Write-Host "Slicing complete: $output"
```

Troubleshooting

Problem	Cause	Solution
First layer not sticking	Bed not level	Bed leveling procedure in printer manual
Supports everywhere	No support type selected	Change to ”Linear” or ”Grid”
Nozzle drags through model	Z-offset too low	Raise Z-offset +0.1mm
Oozing strings between parts	Temp too high	Lower nozzle temp 5-10C
Print breaks mid-way	Adhesion problem	Add brim; check bed level

2. Bambu Studio (Bambu Lab)

Overview

- Developer: Bambu Lab (printer manufacturer)

- Platforms: Windows, Mac, Linux
- Best For: Modern X1-series printers, excellent speed, AMS support
- Download: https://bambulab.com/en/download/studio
- Accessibility: Good contrast, keyboard navigation

Quick Setup Step 1: Create Account & Connect Printer

1. Launch Bambu Studio
2. Sign in with Bambu Lab account
3. Select printer from network
4. Studio auto-detects printer settings

Step 2: Load Model & Configure

1. Drag STL into workspace
2. Auto-arranges on bed plate
3. Default profile applied automatically

Step 3: Key Settings (Bambu-Specific)

Setting	Default	Adjustment	Why
AMS Multi-Color	Off	On (if AMS attached)	Auto-switch filament
Auto-Leveling	Enabled	Keep On	Bambu feature-very reliable
Nozzle Temp	220C	Keep unless specified	Bambu-optimized
Layer Height	0.2mm	0.15mm for detail	Balance speed/quality
Bed Temp	65C	60C for PLA	Standard adhesion

Step 4: Send to Printer

1. Click ”Prepare” (bottom right)
2. Review preview
3. Click ”Send to Device”
4. Printer receives over WiFi
5. Start print from printer screen

Accessible Parameter Explanations Auto-Leveling

- Bambu printers automatically level the nozzle before every print
- Explanation: Saves manual calibration; extremely reliable
- For VI users: Provides confidence that bed is properly prepared

Filament Calibration

- Before first use of new filament color, run ”Filament Calibration”

- This optimizes temperature and speed for that specific filament
- Explanation: Ensures consistent color and strength

Multi-Material (AMS)

- If you have Auto Material System (AMS):
  - Load up to 4 filament colors
  - Studio auto-switches during print
  - Explanation: Multi-color prints without manual intervention

Command-Line Usage (PowerShell)

```
# Slice with Bambu Studio (command-line interface)
$model = "C:\Models\bracelet_holder.stl"
$output = "C:\GCode\bracelet_holder.3mf" # Bambu uses .3mf format
# Bambu Studio CLI
& "C:\Program Files\BambuStudio\bambu-studio.exe" `
   --output "$output" `
   "$model"
Write-Host "Slice saved: $output"
# Send to printer directly
# (Requires API key-see Bambu documentation)
```

Troubleshooting

Problem	Cause	Solution
WiFi not connecting	Network issue	Restart printer WiFi; check SSID
AMS not switching	Filament not detected	Load filament into AMS; recalibrate
Print quality inconsistent	Wrong filament type	Run filament calibration
Nozzle crashes on first layer	Auto-level failed	Manually check nozzle height

3. Cura (Ultimaker)

Overview

- Developer: Ultimaker (Dutch company, open-source)
- Platforms: Windows, Mac, Linux
- Best For: Broad printer support, user-friendly, good documentation
- Download: https://ultimaker.com/software/ultimaker-cura
- Accessibility: Clear UI, good contrast

Quick Setup Step 1: Add Your Printer

1. Launch Cura
2. Go to ”Settings” (top-right)
3. Click ”Printers” -> ”Add Printer”

4. Select your printer model from list
5. Confirm network connection

Step 2: Load & Prepare Model

1. Drag STL into workspace
2. Model auto-scales if needed (confirm size)
3. Right-click -> ”Support” (if overhangs need support)

Step 3: Recommended Settings

Setting	Value	Notes
Profile	Standard	Good balance for most prints
Layer Height	0.2mm	Default; change to 0.15mm for detail
Infill	20%	100% waste for solid parts
Support Angle	50	Auto-generates support for overhangs
Build Plate Adhesion	Brim	Helps first layer stick
Nozzle Temp	200C	Standard PLA
Bed Temp	60C	Standard PLA

Step 4: Print

1. Click ”Slice” (bottom right)
2. Review layer-by-layer preview
3. Click ”Print Over Network” or ”Print to File”
4. Model sends to printer or saves as .gcode

Accessible Settings Explanation Combing Mode

- Off: Nozzle retracts on every travel (quality)
- All: Never retracts (faster, possible stringing)
- Not in Skin: Smart compromise
- For VI users: Retraction prevents nozzle oozing on visible surfaces

Z-Offset (Z Clearance)

- Adjusts first-layer distance
- Too low -> nozzle scrapes bed (bad)
- Too high -> filament doesn't stick (bad)
- Correct -> thin line of plastic sticks to bed

Gradual Infill

- Automatically reduces infill strength away from surface
- Saves filament while maintaining strength
- Explanation: The core doesn't need to be solid

Command-Line Usage (PowerShell)

```
# Cura engine CLI (CuraEngine)
$model = "C:\Models\bracelet_holder.stl"
$output = "C:\GCode\bracelet_holder.gcode"
$config = "default.cfg"
# Requires Cura to be installed; command-line slicing
& "C:\Program Files\Ultimaker Cura\CuraEngine.exe" `
    -c "$config" `
    -o "$output" `
    "$model"
Write-Host "Sliced: $output"
```

Troubleshooting

Problem	Cause	Solution
Model appears too small on bed	Scale wrong	Right-click -> Scale to fit
Stringing between parts	Retraction disabled	Enable retraction in settings
Support doesn't generate	Auto-support off	Enable ”Generate Support”
Printer not found	Network/USB issue	Check connection; restart Cura

4. SuperSlicer (Modification of Prusa)

Overview

- Developer: Community fork of PrusaSlicer
- Platforms: Windows, Mac, Linux
- Best For: Advanced users wanting more control than Prusa offers
- Download: https://github.com/supermerill/SuperSlicer
- Accessibility: Similar to Prusa, more advanced options

Key Differences from Prusa

Feature	PrusaSlicer	SuperSlicer
Arachne Engine	No	Yes-better edges
Seam Position	Limited options	Full control
Pressure Equalization	No	Yes-better bridging
Stealth Mode	No	Yes-quieter/higher quality

When to Use SuperSlicer

- Printing challenging geometries with tight tolerances
- Need advanced surface finish control
- Familiar with PrusaSlicer already and want more power

Quick Start

1. Download SuperSlicer from GitHub
2. Export profile from PrusaSlicer (if you have it)
3. Import into SuperSlicer
4. All settings are compatible with PrusaSlicer

Advanced Settings for SuperSlicer Arachne Engine

- Enables finer edges on walls
- Results in cleaner, more accurate prints
- Takes slightly longer to slice but worth it

Seam Positioning

- Random: Hides seams (good for aesthetic)
- Aligned: Consistent location (good for debugging)
- Rear: Always at back (recommended)

Pressure Equalization

- Helps with bridging (printing across gaps)
- Reduces sagging on overhangs
- Recommended: Enable for complex designs

5. OrcaSlicer (Modern Bamboo Alternative)

Overview

- Developer: Community (independent open-source)
- Platforms: Windows, Mac, Linux
- Best For: Users who want Bambu features without Bambu printer
- Download: https://github.com/SoftFever/OrcaSlicer
- Accessibility: Modern UI, good keyboard support

Why Orca? OrcaSlicer brings Bambu Studio's best features to any printer:

- Excellent defaults
- Fast slicing
- Good preview
- Modern interface

Quick Setup

1. Download OrcaSlicer
2. Select your printer (not just Bambu models)
3. Load STL
4. Uses good defaults-usually ready to print

Key Settings

Setting	Value	Why
Wall Loops	2	Strong walls, visible detail
Internal Solid Layer	4	Strength for brackets/connectors
Infill Pattern	Grid	Balanced strength
Line Width	Auto	Matches nozzle diameter
Speed	80 mm/s	Balanced quality/speed

Accessible Features Filament Manager

- Track filament type, color, weight used
- Explanation: Know when to buy new filament

Print Time Estimation

- Accurate prediction of print duration
- Updates during slicing

Material Presets

- Pre-configured settings for common filaments
- Just select material type; settings auto-apply

6. IdeaMaker (Raise3D)

Overview

- Developer: Raise3D
- Platforms: Windows, Mac, Linux
- Best For: Raise3D printer users; advanced features
- Download: https://www.raise3d.com/ideamaker
- Accessibility: Professional UI, detailed settings

When to Use

- Using a Raise3D printer (excellent multi-nozzle support)
- Need industrial-strength slicing
- Want dual-extrusion (2-color) printing

Quick Start

1. Launch IdeaMaker
2. Add printer (Raise3D models have built-in profiles)
3. Load STL
4. Adjust layer height and infill
5. Slice and send to printer

Key Features Dual Extrusion

- Two nozzles = two colors in one print
- Useful for: Bracelets (core + colored rim)
- Requires: Coordinating two materials

Advanced Support

- Tree support (uses less material)
- Grid support (stronger for larger parts)

Print Balancing

- Optimizes nozzle movement for efficiency
- Reduces print time without sacrificing quality

7. Fusion 360 (CAD + Slicer)

Overview

- Developer: Autodesk
- Platforms: Windows, Mac
- Best For: If you already use Fusion 360 for CAD
- Download: https://www.autodesk.com/products/fusion-360
- Accessibility: Integrated environment; good for learning CAD+Slicing together

Why Integrate CAD + Slicing? Traditional workflow:

```
Design in CAD -> Export to STL -> Open in Slicer -> Slice -> Print
```

Fusion 360 workflow:

```
Design in Fusion -> Run Print Preparation -> Slice -> Print
```

Advantage: Stay in one program; no format conversion

Quick Start

1. Design in Fusion 360 (or import STL)
2. Select part
3. Go to "3D Print" tab
4. Click "Print Preparation"
5. View auto-generated support and alignment
6. Adjust layer height, infill, etc.
7. Slice and export G-code

Integration with 3dMake If using Fusion 360 for CAD:

1. Export SCAD design to STL
2. 3dm export-stl src/main.scad output/main.stl
3. Then import into Fusion 360
4. For more complex designs that need refinement

Universal Troubleshooting Guide

Problem	Diagnosis	Solution
Print won't stick to bed	Bed temperature too low	Raise bed temp +5C; check bed level
	Bed not level	Manual leveling procedure (printer manual)
	Build plate dirty	Clean with isopropyl alcohol
Nozzle hits model mid-print	Z-offset wrong	Adjust Z-offset; re-level bed
	Model placed too low on bed	Use ”Arrange on Bed” tool in slicer
Supports won't remove	Too much support generated	Reduce support density or angle
	Support too strong	Reduce support material percentage
Stringy/Oozing	Nozzle too hot	Reduce temp by 5-10C
	Retraction disabled	Enable retraction in settings
	Travel speed too fast	Reduce travel speed
Layer shifting (X/Y)	Belt tension off	Check belt tension (printer manual)
	Stepper motor power issue	Firmware issue-check printer logs
Model prints poorly but slices look good	Filament quality issue	Try different filament batch
	Nozzle clogged	Unclog nozzle (heat -> purge -> clean)

Recommended Slicer for Each Situation

Scenario	Best Slicer	Why
I'm a beginner	PrusaSlicer	Clear defaults, excellent UI
I have a Bambu printer	Bambu Studio	Native, best features
I have an Ultimaker	Cura	Official support, broad compatibility

Scenario	Best Slicer	Why
I have a Raise3D	IdeaMaker	Official, dual-extrusion support
I want advanced control	SuperSlicer	Maximum customization
I want modern/fast slicing	OrcaSlicer	Great defaults, any printer
I use Fusion 360 for CAD	Fusion 360 Print Prep	Integrated workflow

PowerShell Integration: Batch Slicing

Slice Multiple Files Automatically

```
# Batch slice all SCAD designs in a project
$scadDir = "C:\Projects\3dMake\src"
$slicerConfig = "default.cfg"
$outputDir = "C:\Projects\3dMake\gcode"
# Find all SCAD files
$scadFiles = Get-ChildItem -Path $scadDir -Filter "*.scad" -Recurse
foreach ($scadFile in $scadFiles) {
    $stlFile = $scadFile.FullName -replace ".scad$", ".stl"
    $gcodeFile = Join-Path $outputDir ($scadFile.BaseName + ".gcode")
    # Export SCAD to STL
    Write-Host "Exporting: $($scadFile.Name)"
    & "C:\Program Files\OpenSCAD\openscad.exe" -o "$stlFile" "$($scadFile. →
        ↪ FullName)"
    # Slice STL
    Write-Host "Slicing: $($scadFile.Name)"
    & "C:\Program Files\Prusa3D\PrusaSlicer\prusa-slicer.exe" `
        --load-config-file "$slicerConfig" `
        --export-gcode "$gcodeFile" `
        "$stlFile"
    Write-Host "Complete: $gcodeFile"
}
Write-Host "Batch slicing finished."
```

Monitor Printing Progress

```
# Connect to printer API and monitor print status
# (Requires printer to support API-check documentation)
$printerIP = "192.168.1.100" # Your printer's IP
$printerPort = 8080 # Typical API port
# Get current print status
$status = Invoke-WebRequest -Uri "[https://example.com](https://example.com) →
    ↪ `
    -UseBasicParsing | ConvertFrom-Json
Write-Host "State: $($status.state.text)"
Write-Host "Progress: $($status.progress.completion)%"
Write-Host "Time remaining: $($status.progress.printTimeLeft) seconds"
```

Accessibility Best Practices for Slicing

Describing Sliced Models

```
# Use 3dm describe to understand the slicer's output
# (Best practice: describe after slicing to verify settings)
3dm describe output/bracelet_holder.stl
# Output includes:
# - Number of parts
# - Dimensions
# - Surface area
# - Volume
```

Testing Sliced Parts

1. Weight Check:
    - Calculate expected weight from volume + material density
    - Compare to actual printed weight
    - Indicates if infill is correct
2. Dimensional Check:
    - Use calipers to verify critical dimensions
    - Compare to SCAD parameters
    - Check tolerance stack-up
3. Functional Test:
    - Assemble with other parts
    - Test strength by loading with known weight
    - Verify support removal didn't damage part

Summary

This appendix provides:

- [YES] 7 slicer workflows, settings, and troubleshooting
- [YES] Accessible parameter explanations
- [YES] Command-line integration for PowerShell
- [YES] Comparison table for choosing a slicer
- [YES] Batch processing automation examples
- [YES] Accessibility best practices

Use this guide whenever you:

- Start a new print
- Switch slicers
- Encounter quality issues
- Want to automate slicing workflows
- Need troubleshooting help

Appendix B: Material Properties & Selection Guide

This appendix covers material properties, characteristics, and selection criteria for 3D printing filaments. Each material includes:

- Physical properties (strength, flexibility, temperature tolerance)
- Printing parameters (nozzle temp, bed temp, speed)
- Suitability for different projects
- Accessibility considerations (measurement-based verification)
- Cost/availability comparison

Referenced in: Lessons 5 (Safety), 6-10 (Projects), 11 (Customer Requirements)

Overview: Why Material Matters

The material you choose affects:

- Strength: Can the part hold weight?
- Flexibility: Will it bend or break?
- Temperature: Can it withstand heat?
- Durability: Will it last months or years?
- Cost: Budget constraints?
- Printability: Ease of use for beginners?
- Appearance: Texture, color, finish?

The Material Selection Flowchart

```
What's the primary requirement?
+-- Strength & Detail?
    +-- PLA (best for beginners, detail)
        or PETG (stronger, tougher)
+-- Flexibility?
    +-- TPU/TPE (rubber-like)
+-- Heat Resistance?
    +-- ABS or Polycarbonate
+-- Transparency?
    +-- PETG or Polycarbonate
+-- Food Contact?
    +-- FDA-approved PETG or PLA
+-- Cost-Conscious?
    +-- PLA (cheapest, easiest)
```

1. PLA (Polylactic Acid)

Properties at a Glance

Property	Rating	Notes
Strength	[3/5]	Good for most projects; not for dynamic loads
Flexibility	[1/5]	Brittle; will snap under stress
Temperature Resistance	[2/5]	Softens around 60C; bad for hot environments
Ease of Printing	[5/5]	Most forgiving; best for beginners
Cost	[5/5]	Cheapest option (~$20/kg)
Appearance	[5/5]	Excellent surface finish; many colors
Availability	[5/5]	Available everywhere

Ideal Projects

- [YES] Decorative pieces (jewelry, miniatures)
- [YES] Enclosures/shells (light loads)
- [YES] Prototype/mockups
- [YES] Low-stress connector clips
- [YES] Educational demonstrations
- [NO] Load-bearing brackets

- [NO] Hinges or flexing parts
- [NO] Heat-resistant applications

Printing Parameters

Parameter	Value	Tolerance
Nozzle Temp	200C	190-210C (varies by brand)
Bed Temp	60C	50-65C
Print Speed	50 mm/s	40-60 mm/s
Retraction	5mm @ 40 mm/s	Yes, prevents stringing
Cooling Fan	100%	High cooling improves quality
First Layer	Slower (25 mm/s)	Ensures adhesion
Layer Height	0.2mm	0.1-0.3mm depending on detail

PLA Variants Standard PLA

- Most common, reliable
- Best for beginners

PLA Pro / Enhanced PLA

- Slightly stronger than standard PLA
- Same temperature parameters
- ~10% cost premium

Silk PLA

- Glossy finish instead of matte
- Same strength as standard PLA
- Slightly slower to print

Marble or Color-Changing PLA

- Visual effects
- Same printing parameters
- Mostly for aesthetics

Common Issues & Solutions

Problem	Cause	Solution
Warping on corners	Bed too hot or cooling too fast	Reduce bed temp to 50C; disable cooling for first layer
Stringing	Temp too high	Lower nozzle temp 5C
Poor layer adhesion	Nozzle too high	Lower Z-offset 0.1mm

Problem	Cause	Solution
Brittleness after print	Normal for PLA	Not a problem; expected behavior
Nozzle clogs on retraction	Temperature inconsistency	Ensure stable nozzle temp +/- 5C

Why PLA for This Course PLA is recommended for all Lessons 1-11 because:

1. Beginner-friendly: Most forgiving material
2. Predictable: Consistent across different printers
3. Cost-effective: Maximize printing volume on student budget
4. Accessibility: Easier to troubleshoot for new users
5. Safe: Non-toxic, low fume emission
6. Available: Found at any 3D printing supplier

2. PETG (Polyethylene Terephthalate Glycol)

Properties at a Glance

Property	Rating	Notes
Strength	[4/5]	Tougher than PLA; better for load-bearing
Flexibility	[2/5]	Better than PLA but still primarily rigid
Temperature Resistance	[3/5]	Softens around 85C; better than PLA
Ease of Printing	[4/5]	Slightly more challenging than PLA
Cost	[4/5]	~$25/kg (slightly more than PLA)
Appearance	[4/5]	Good surface finish; slightly glossier than PLA
Availability	[5/5]	Widely available

When to Use PETG Instead of PLA

- Functional brackets or mounts (where strength matters)
- Parts that may be load-bearing (shelves, holders)
- Outdoors/higher temperature environments
- Parts requiring transparency (clear PETG available)
- Better moisture resistance (vs. PLA)

Printing Parameters

Parameter	Value	Tolerance
Nozzle Temp	235C	230-245C

Parameter	Value	Tolerance
Bed Temp	80C	75-85C
Print Speed	50 mm/s	40-60 mm/s (same as PLA)
Retraction	4mm @ 40 mm/s	Slightly less than PLA
Cooling Fan	30-50%	Less cooling than PLA
First Layer	Normal (50 mm/s)	Harder to adjust than PLA
Layer Height	0.2mm	0.1-0.3mm

Comparison: PETG vs. PLA

Aspect	PLA	PETG
Nozzle Temp	200C	235C
Bed Temp	60C	80C
Strength	Good	Better (20% stronger)
Flexibility	Brittle	More resilient
Heat Tolerance	60C	85C
Ease	Very easy	Easy (needs tweaking)
Cost	$20/kg	$25/kg
Outdoor Use	Not ideal	Better

Projects for PETG

- [YES] Phone stand (needs strength)
- [YES] Bracket or shelf support
- [YES] Flexible clip (needs resilience)
- [YES] Outdoor item
- [YES] Clear enclosure (if using clear PETG)
- [NO] Fine detail work (slightly courser finish)
- [NO] Food-contact items (not food-safe unless specified)

3. ABS (Acrylonitrile Butadiene Styrene)

Properties at a Glance

Property	Rating	Notes
Strength	[4/5]	Similar to PETG; good impact resistance
Flexibility	[3/5]	More flexible than PETG
Temperature Resistance	[5/5]	Softens around 105C; best of common materials
Ease of Printing	[2/5]	Requires enclosure/heated bed; challenging
Cost	[4/5]	~$25-30/kg

Property	Rating	Notes
Appearance	[3/5]	Rougher than PLA; requires post-processing
Availability	[4/5]	Good availability; not as universal as PLA

When to Use ABS

- High-temperature environments (near heat sources)
- Mechanical parts (gears, bearings)
- Durability (parts lasting years)
- Post-processing (can be sanded, glued, vapor-smoothed)
- Professional applications (not toys/decorative)

Why ABS is Challenging ABS requires:

1. Enclosed environment: Minimize temperature fluctuations
2. Heated bed: 100C+ (much higher than PLA)
3. Controlled cooling: Too-fast cooling causes warping
4. Ventilation: ABS emits fumes (acetone-like smell)

Printing Parameters

Parameter	Value	Tolerance
Nozzle Temp	240C	230-250C
Bed Temp	100C	95-105C
Enclosure	Required	Maintains heat (reduces warping)
Print Speed	40 mm/s	Slower than PLA
Retraction	3mm @ 30 mm/s	Very short
Cooling Fan	0%	OFF (causes warping)
First Layer	Slow (30 mm/s)	Critical for adhesion

Projects for ABS

- [YES] Mechanical components
- [YES] Heat-resistant enclosure
- [YES] Durable outdoor item
- [YES] Professional prototypes
- [NO] Beginner projects (too challenging)
- [NO] Decorative/aesthetic work (not recommended)
- [NO] Flexible parts

Accessibility Note ABS is not recommended for this course because:

1. Requires enclosed printer (expensive for beginners)
2. High failure rate for inexperienced users
3. Strong odor (ventilation concerns for some users)
4. Requires extra equipment (acetone for post-processing)

4. TPU / TPE (Thermoplastic Polyurethane / Elastomer)

Properties at a Glance

Property	Rating	Notes
Strength	[3/5]	Good; impact-resistant
Flexibility	[5/5]	Very flexible; rubber-like
Temperature Resistance	[3/5]	Softens around 80C; moderate
Ease of Printing	[3/5]	Needs tweaking; flexible materials are tricky
Cost	[3/5]	~$30-40/kg (expensive)
Appearance	[4/5]	Smooth; feels good tactilely
Availability	[4/5]	Growing availability

What is TPU? TPU is a flexible rubber-like plastic that:

- Doesn't crack when bent
- Absorbs impact
- Returns to original shape
- Bridges the gap between plastic and rubber

Printing Parameters

Parameter	Value	Tolerance
Nozzle Temp	215C	210-225C
Bed Temp	60C	50-70C
Print Speed	20-30 mm/s	VERY SLOW (flexibility needs time)
Retraction	Minimal or Off	0-1mm (flexible material doesn't retract well)
Cooling Fan	0%	Off (material needs heat)
Line Width	0.5mm	Wider than normal (flexible material bridges)

Projects for TPU

- [YES] Phone case (needs flexibility + protection)
- [YES] Flex joints / hinges

- [YES] Gasket or seal
- [YES] Shoe insert or orthotic
- [YES] Tactile button (for accessibility)
- [NO] Fine detail work (too stretchy)
- [NO] Decorative items (usually not aesthetic)
- [NO] Precision parts

Challenges with TPU

1. Very slow printing: 5-10x slower than PLA
2. Stringing: Flexible material tends to ooze
3. Flexible bed needed: Standard beds may not work
4. Post-processing difficult: Hard to sand/glue

Why TPU for Accessibility TPU is excellent for accessibility because:

- Can create tactile buttons/indicators
- Flexible grips for ergonomic handles
- Gaskets that don't damage delicate equipment
- Accessible because: Achievable by all users if given proper guidance

5. Polycarbonate (PC)

Properties at a Glance

Property	Rating	Notes
Strength	[5/5]	Extremely strong; impact-resistant
Flexibility	[2/5]	Rigid; similar to PETG
Temperature Resistance	[5/5]	Best; softens around 130C
Ease of Printing	[2/5]	Difficult; prone to warping
Cost	[2/5]	Most expensive (~$50+/kg)
Appearance	[4/5]	Transparent/translucent options
Availability	[2/5]	Limited; specialty supply

When to Use Polycarbonate

- Highest strength required
- Transparent/bullet-proof enclosure needed
- Extreme temperature environment
- Professional/industrial applications

Why Polycarbonate is Not for This Course

1. Extreme difficulty (high failure rate)
2. Very expensive (3-5x cost of PLA)

3. Requires industrial-grade printer
4. Post-processing complex

6. Nylon (PA)

Properties at a Glance

Property	Rating	Notes
Strength	[5/5]	Very strong; can be flexible
Flexibility	[4/5]	More flexible than ABS
Temperature Resistance	[4/5]	Good; softens around 120C
Ease of Printing	[2/5]	Difficult; very temperature-sensitive
Cost	[3/5]	~$30-40/kg
Appearance	[3/5]	Matte finish; less aesthetic than PLA
Availability	[3/5]	Growing but limited

Nylon Use Cases

- [YES] Mechanical parts (gears, hinges)
- [YES] Flexible connectors
- [YES] Threads/screws
- [YES] High-stress applications
- [NO] Not for beginners

Material Comparison Table

Material	Nozzle	Bed	Strength	Flexibility	Heat	Ease	Cost	Best For
PLA	200C	60C	[3/5]	[1/5]	[2/5]	[5/5]	$	Beginners, detail
PETG	235C	80C	[4/5]	[2/5]	[3/5]	[4/5]	$$	Functional parts
ABS	240C	100C	[4/5]	[3/5]	[5/5]	[2/5]	$$	High-temp/mechanical
TPU	215C	60C	[3/5]	[5/5]	[3/5]	[3/5]	$$$	Flexibility/tactile
PC	280C	110C	[5/5]	[2/5]	[5/5]	[1/5]	$$$$	Extreme strength
Nylon	250C	85C	[5/5]	[4/5]	[4/5]	[2/5]	$$$	Mechanical/flexible

Filament Quality Factors

Why Not All PLA is the Same Diameter Tolerance

- Good filament: +/- 0.03mm
- Poor filament: +/- 0.1mm or worse
- Impact: Inconsistent extrusion, layer quality varies

Dryness

- PLA absorbs moisture from air
- Wet filament = weak prints + bubbles
- Solution: Store in sealed container with desiccant

Color Consistency

- Good brands: Same color throughout
- Poor brands: Color varies spool-to-spool

Impurities

- Good: Minimal impurities
- Poor: Visible specs/contaminants -> possible clogs

How to Check Filament Quality (Non-Visually)

1. Weight Check

```
Known: 1kg spool
Weigh spool + filament
Calculate remaining filament
Should match spool markings
```

2. Diameter Check

```
Use caliper to measure multiple points
Should be consistent +/- 0.03mm
If varying, likely lower quality
```

3. Dryness Test

```
Feel texture: Should be smooth, not tacky
Smell: Should be neutral (not musty)
If wet, store in sealed container with desiccant
```

4. Print Test

```
Print small cube (20mm x 20mm x 20mm)
Inspect surface: Smooth or bubbly?
Weight: Does it match expected weight?
```

Storage & Maintenance

Proper Filament Storage Temperature

- Store between 15-25C
- Avoid direct sunlight (fades color, degrades material)

Humidity

- Keep below 40% humidity
- Use desiccant packets in sealed containers
- Change desiccant every 2-3 months

Organization

- Label spools with: Material, color, date opened, approx. remaining
- Store vertically or on spindle (prevents kinking)

Filament Degradation Signs

Sign	Cause	Solution
Weak prints / breaking easily	Filament aged or wet	Replace with fresh filament
Discoloration or spots	Oxidation or contamination	Not usable; discard safely
Brittle or crumbly	Overheated or UV damage	Not usable; discard
Slight fading (color)	UV exposure	Still usable; just faded

Cost Analysis

Cost per Project

```
Material cost = (Filament weight used) x (Cost per kg)

Example: Bracelet holder
- Weight: 25 grams
- Material: PLA at $20/kg
- Cost: (25g / 1000g) x $20 = $0.50

vs. PETG at $25/kg: $0.625
vs. ABS at $30/kg: $0.75
```

Budget Tips

1. Buy bulk: 5kg spool is cheaper per gram than 1kg
2. Buy sales: 30-40% discounts common during sales
3. Brand matters: Premium brands slightly more but more reliable
4. Generic brands: Often acceptable if reviews are good

Recommended Brands (Ranked by Beginner-Friendliness)

Rank	Brand	Known For	Cost	Notes
1	Prusament	Reliability, Prusa compatibility	$$$	Best; excellent support
2	Matter-Hackers	Quality, variety	$$	Very good; educational focus
3	Fillamen-tum	European quality	$$	Excellent; eco-friendly
4	Overture	Value, consistency	$	Good budget option
5	eSUN	Variety, affordable	$	Decent; variable quality

Material Selection Decision Tree

```
START HERE: What's most important?

GOAL: Beginner success?
  YES - PLA (best choice)
  NO - Next question

GOAL: Strength matters?
  YES - PETG or Nylon
  NO - Next question

GOAL: Flexibility needed?
  YES - TPU (rubber-like)
  NO - Next question

GOAL: High temperature?
  YES - ABS or Polycarbonate
  NO - Next question

GOAL: Transparent?
  YES - Clear PETG or Polycarbonate
  NO - Use PLA

FINAL CHOICE:
- If unsure, use PLA
- If needs strength, use PETG
- If needs flexibility, use TPU
- If needs heat, use ABS
```

PowerShell Integration: Track Material Usage

```powershell
# Track filament usage across all projects
$materialLog = @"
ProjectName,Material,ColorName,WeightUsed(g),DatePrinted,Notes
"@
# Example entries
$materialLog += "`nBraceletHolder,PLA,NaturalWhite,25.3,2024-01-15,Final →
    ↪ design"
$materialLog += "`nPhoneStand,PETG,Black,47.2,2024-01-16,Needs strength"
$materialLog += "`nKeycap,PLA,Red,12.5,2024-01-17,Prototype"
# Save to CSV
$materialLog | Out-File "C:\Projects\material-log.csv"
# Analyze usage
$materials = Import-Csv "C:\Projects\material-log.csv"
$totalWeight = ($materials | Measure-Object -Property "WeightUsed(g)" -Sum). →
    ↪ Sum
$avgWeight = ($materials | Measure-Object -Property "WeightUsed(g)" -Average →
    ↪ ).Average
Write-Host "Total weight used: $totalWeight grams"
```

```
Write-Host "Average per project: $avgWeight grams"
# Calculate cost (PLA at $20/kg)
$costPerKg = 20
$totalCost = ($totalWeight / 1000) * $costPerKm
Write-Host "Estimated cost: $$($totalCost.ToString("F2"))"
```

Summary

Key Takeaways:

1. Start with PLA: Best for learning; most forgiving
2. Understand the tradeoffs: Strength vs. ease, cost vs. quality
3. Match material to project: Decorative = PLA; functional = PETG
4. Store properly: Desiccant, cool, dark location
5. Track usage: Know what works for future projects
6. Test before committing: Print small test on new filament

Recommended Progression:

- Lessons 6-7: PLA (simplest)
- Lessons 8-9: PLA or PETG (functional parts)
- Lesson 10: PETG or TPU (testing materials)
- Lesson 11: Student choice (depends on stakeholder requirements)

Appendix C: Tolerance Testing & Quality Assurance Matrix

This appendix provides measurement-based testing methodology for verifying that 3D-printed parts meet design specifications. It's designed to be used non-visually-focusing on calipers, scales, and functional tests rather than visual inspection.

Referenced in: Lessons 8-10 (Complex Design, Troubleshooting, Mastery)

Overview: What is Tolerance?

Tolerance is the acceptable range of variation in dimensions:

```
Design spec: Hole diameter = 6mm
Tolerance: +/-0.5mm
Acceptable range: 5.5mm to 6.5mm
Actual print: 5.8mm [YES] (within tolerance)
or 7.2mm [NO] (exceeds tolerance)
```

Why Tolerance Matters

1. Assembly: Parts must fit together
2. Function: Fit too tight = stuck; too loose = falls apart
3. Safety: Wrong tolerance = part failure
4. Cost: Tight tolerance = slower, more waste

Essential Measurement Tools

1. Digital Calipers What they measure:

- Outside diameter (part width)
- Inside diameter (hole width)
- Depth

How to use non-visually:

1. Gently close calipers until they barely touch the part
2. Feel the resistance (should be light, not forced)
3. Read digital display with audio feedback or manually
4. Record three measurements at different locations
5. Average the three measurements

Accuracy: +/-0.05mm (very precise for 3D printing)

2. Digital Scale (Kitchen Scale) What it measures:

- Part weight (indicates infill, material type)

How to use non-visually:

1. Place part on scale
2. Wait for reading to stabilize (1-2 seconds)
3. Read digital display (in grams)
4. Compare to expected weight

Accuracy: +/-1g (good enough for verification)

Why it matters:

- Too light = infill too low or void inside part
- Expected weight = indicates proper slicing

3. Test Jig / Go/No-Go Gauge What it measures:

- Pass/fail tolerance testing without calipers

How to make:

```
// Go/No-Go gauge for bracelet peg holes
// Tests if hole is within acceptable range
pegdiameter = 6; // Design spec
tolerance = 0.5; // Tolerance
godiameter = pegdiameter - tolerance; // Min acceptable (5.5)
nogodiameter = pegdiameter + tolerance; // Max acceptable (6.5)
module gogauge() {
  // Part should fit through this easily
  cylinder(h=10, r=godiameter/2);
}
module nogogauge() {
  // Part should NOT fit through this
  cylinder(h=10, r=nogodiameter/2);
}
// Test jig with both gauges
```

```
union() {
  translate([0, 0, 0]) gogauge();
  translate([0, 15, 0]) nogogauge();
}
```

Use non-visually:

- Try inserting peg into go-gauge -> Should slide easily
- Try inserting peg into no-go-gauge -> Should NOT fit
- If both tests pass = tolerance correct

Quality Assurance Testing Matrix

Critical Dimensions to Test Create a Test Plan before printing:

Part	Dimension	Spec	Tolerance	How to Test
Bracelet Holder	Base width	127mm	+/-2mm	Measure with calipers (multiple points)
	Peg diameter	6mm	+/-0.5mm	Test fit with go/no-go gauge
	Peg spacing	8mm	+/-1mm	Measure distance between pegs
	Back wall height	120mm	+/-2mm	Measure with calipers
Phone Stand	Slope angle	20	+/-3	Calculate from height/depth ratio
	Weight capacity	200g	150-250g	Load test (see below)
	Stability	N/A	Pass/fail	1-hour load test without tipping

Pre-Print Planning Before slicing, define:

1. Critical dimensions: Which measurements matter most?
2. Acceptable range: What tolerance is realistic?
3. Test method: How will you verify?
4. Pass/fail criteria: What does "success" look like?

Example Plan for Phone Stand:

```
# Phone Stand - Quality Assurance Plan

## Critical Dimensions
1. Slope angle: 20 +/-3
2. Base width: 80mm +/-1mm
3. Stand height: 60mm +/-2mm

## Functional Tests
1. Stability: Hold 200g for 1 hour without tipping
2. Grip: Phone doesn't slide during tilt
```

```
3. Assembly: Back brace attaches without force

## Pass/Fail Criteria
[YES] PASS if:
- All dimensions within tolerance
- Phone holds weight for 1 hour
- No cracks or layer separation

[NO] FAIL if:
- Any dimension >2mm off spec
- Phone slides or part tips
- Visible cracks
```

Measurement Procedures

Procedure 1: Linear Dimension (Width, Height, Depth) Equipment needed: Digital calipers

Steps:

1. Place part on flat surface
2. Position caliper jaws perpendicular to surface
3. Gently close jaws until they just touch part
4. Feel for light resistance (not forced)
5. Read digital display
6. Record measurement
7. Repeat at 3 different locations
8. Average the three readings
9. Compare to design spec +/- tolerance

Example:

```
Design spec: 127mm (bracelet holder width)
Tolerance: +/-2mm (acceptable: 125-129mm)
Measurements:
  Location 1: 126.8mm
  Location 2: 127.1mm
  Location 3: 126.5mm
  Average: 126.8mm [YES] (within tolerance)
```

Procedure 2: Hole or Peg Diameter Equipment needed: Digital calipers, Go/No-Go gauges (optional)

Method A: Direct Measurement

1. Insert caliper jaws into hole/around peg
2. Adjust jaws to gently touch surfaces
3. Feel for light resistance on both sides
4. Read digital display (inside measurement mode)
5. Record three measurements
6. Average the readings

Method B: Go/No-Go Gauge

1. Print test jigs (go & no-go gauges)

2. Attempt to insert peg/hole into go-gauge -> Should slide through easily with light resistance
3. Attempt to insert peg/hole into no-go-gauge -> Should NOT fit or fit with visible resistance
4. If both pass -> dimension is acceptable

Example: Peg Diameter

Design spec: 6.0mm Tolerance: +/-0.5mm (acceptable: 5.5-6.5mm) Method A (Direct): Measurement 1: 5.9mm Measurement 2: 6.0mm Measurement 3: 5.8mm Average: 5.9mm [YES] Method B (Go/No-Go): Slides through go-gauge (5.5mm) -> [YES] Doesn't fit no-go-gauge (6.5mm) -> [YES] Result: PASS

Procedure 3: Surface Finish / Layer Quality Equipment needed: Caliper, ruler, touch/texture assessment

Step 1: Surface Texture

Run fingers/hand over surface: [YES] Smooth -> Good quality Slightly rough -> Acceptable [NO] Very rough/bumpy -> Quality issue

Step 2: Layer Line Visibility

Feel horizontal ridges (layer lines): [YES] Barely perceptible -> Good (0.2mm layers) Noticeable but even -> OK (0.25mm layers) [NO] Very pronounced -> Quality issue

Step 3: Dimensional Consistency

```
Measure thickness at multiple points:
  Design spec: 3mm wall thickness
  Measure at 5 locations
  Record all measurements
  All should be within +/-0.2mm of each other
  Example:
    Pt 1: 3.0mm
    Pt 2: 3.1mm
    Pt 3: 3.0mm
    Pt 4: 2.9mm
    Pt 5: 3.1mm
  Std Dev: 0.08mm [YES] (very consistent)
```

Procedure 4: Weight Verification (Confirms Infill) Equipment needed: Digital scale

Step 1: Calculate Expected Weight

Expected weight = Volume x Density x Infill% For bracelet holder (PLA): Volume = 127mm x 80mm x 120mm = 1,219,200 mm = 1,219.2 cm PLA density = 1.24 g/cm Infill = 20% Expected weight = 1,219.2 x 1.24 x 0.20 = 302.4g Acceptable range: 290-315g (+/-5%)

Step 2: Measure Actual Weight

1. Place part on digital scale

2. Wait for reading to stabilize (1-2 seconds)
3. Read display in grams
4. Compare to expected weight

Step 3: Interpret Result

Actual Weight	Interpretation
290-315g	[YES] Correct infill, no internal voids
<280g	Infill too low or significant voids
>320g	Infill too high (was it supposed to be 20%?)

Functional Testing

Test 1: Load Testing (Strength) Purpose: Verify part can hold weight without failure

Equipment needed:

- Digital scale
- Test weights (or books, water jugs)
- Calipers (to check for deflection)

Procedure:

1. Measure baseline dimensions (part unloaded) Baseline height: 60mm
2. Place test weight on part Added weight: 200g
3. Wait 5 minutes (allow part to settle)
4. Measure dimensions again New height: 59.8mm Deflection: 0.2mm (acceptable)
5. Observe for cracks (visual or tactile) No cracks: [YES]
6. Remove weight and wait 5 minutes
7. Measure dimensions again (should return to baseline) Height after unload: 60.0mm Recovery: Complete [YES] Result: PASS (part handles load without permanent deformation)

Acceptance Criteria for Phone Stand:

[YES] PASS if:

- Deflection <0.5mm under 200g load
- No cracks visible/felt
- Full recovery after load removed [NO] FAIL if:
- Deflection >1mm
- Visible cracks
- Permanent deformation after unload

Test 2: Assembly Testing Purpose: Verify parts fit together as designed

Equipment needed:

- Calipers
- Go/No-Go gauges

Procedure for Multi-Part Assembly (e.g., Stackable Bins):

Part A (Bin body) dimensions:

- Top opening: 50mm x 50mm +/-1mm
- Wall thickness: 2mm +/-0.2mm Part B (Stacking rim) dimensions:
- Base diameter: 50mm +/-0.5mm
- Should nest inside Part A Test procedure:

1. Measure Part A opening: 50.1mm [YES]
2. Measure Part B base: 49.8mm [YES]
3. Attempt to nest Part B into Part A -> Should fit with light resistance -> Feel for smooth insertion (no catching)
4. Check for rocking (part should sit stable)
5. Apply 500g load (simulate stacking) -> Should hold without slipping

Test 3: Durability Testing (Repeated Use) Purpose: Verify part doesn't fail after repeated cycles

Example: Phone Stand Tilt Cycles

1. Record baseline dimensions
2. Cycle 1: Place phone, tilt to max angle, remove
3. Inspect for cracks or damage
4. Repeat cycle 10 times
5. Measure dimensions again
6. Compare to baseline: Should be <0.1mm change If no damage after 10 cycles -> Durable [YES]

Tolerance Stack-Up (Multi-Part Designs)

When multiple parts are assembled, tolerances add:

Design: Part A opening: 50mm +/-0.5mm Part B base: 50mm +/-0.5mm Worst-case fit: Part A minimum: 49.5mm Part B maximum: 50.5mm Difference: 1.0mm (VERY TIGHT or won't fit!) Solution: Increase tolerance on Part B to +/-0.3mm Part B minimum: 49.7mm Part B maximum: 50.3mm Now fits inside Part A: 49.5-50.5mm [YES]

Tolerance Stack-Up Calculation For N parts in assembly:

```
Total tolerance = (tolerance + tolerance + ... + tolerance)
Example with 3 parts (each +/-0.5mm):
Total = (0.5 + 0.5 + 0.5)
      = 0.75
      = 0.87mm
```

Common Dimensional Problems & Fixes

Problem	Typical Cause	How to Fix	Prevent Next Time
Part too small (all dimensions off)	Scale wrong in slicer	Scale STL up in CAD or slicer	Verify scale before slicing
Holes too small	Compensation shrinkage	Increase hole diameter +0.5mm	Add shrinkage factor to design
Walls too thin	Layer squishing	Check first layer height; may need recalibration	Measure first layer thickness
Infill showing through (loose fill)	Infill too low	Increase infill % to 25-30%	Recalculate for part type
Inconsistent across print	Nozzle clogging mid-print	Clean nozzle; check filament quality	Use quality filament; monitor print
Z-axis dimensions off	Z-axis uncalibrated	Run Z-calibration procedure	Calibrate before critical prints
Dimensions change between prints	Thermal drift	Print in stable temperature	Use enclosed printer if possible

Testing Checklist Template

Print this checklist before starting a new project:

```
# Quality Assurance Checklist: [Project Name]

## Design Specs
- [ ] Part A width: mm +/- mm
- [ ] Part B height: mm +/- mm
- [ ] Hole diameter: mm +/- mm
- [ ] Assembly fit tolerance: mm

## Pre-Print
- [ ] STL exported correctly
- [ ] Scale verified
- [ ] Supports configured
- [ ] Slice file reviewed

## Post-Print (Dimensional)
- [ ] Part A width measured (3 points): mm, mm, mm
  - [ ] Within tolerance? YES/NO
- [ ] Part B height measured (3 points): mm, mm, mm
  - [ ] Within tolerance? YES/NO
- [ ] Hole diameter measured (go/no-go): PASS/FAIL
- [ ] Part weight measured: g (expected: g +/-g)
  - [ ] Within expected weight? YES/NO

## Post-Print (Functional)
- [ ] Load test (if applicable): PASS/FAIL
- [ ] Assembly test: PASS/FAIL
- [ ] Surface quality acceptable: YES/NO
- [ ] No cracks or voids: YES/NO

## Result
- [ ] ACCEPT (all tests pass)
- [ ] REWORK (minor issue, can fix)
- [ ] REJECT (major issue, reprint)

## Notes
[Space for observations]
```

PowerShell Integration: Track Quality Metrics

```
# Track print quality across multiple projects
$qualityLog = @"
ProjectName,Date,Material,PartCount,DimensionsPass,FunctionalPass,Weight(g), →
    ↪ Notes
"@
# Log entries
$qualityLog += "`nBraceletHolder,2024-01-15,PLA,1,PASS,PASS,302.1,Perfect  →
    ↪ fit"
$qualityLog += "`nPhoneStand,2024-01-16,PETG,2,PASS,PASS,487.3,Strong joints →
    ↪ "
$qualityLog += "`nStackableBins,2024-01-17,PLA,3,FAIL,N/A,N/A,Holes too  →
    ↪ small reprint"
# Save to CSV
$qualityLog | Out-File "C:\Projects\quality-log.csv"
# Analyze pass/fail rate
$log = Import-Csv "C:\Projects\quality-log.csv"
$passCount = ($log | Where-Object { $.DimensionsPass -eq "PASS" }).Count
$totalCount = $log.Count
$passRate = ($passCount / $totalCount) * 100
Write-Host "Print success rate: $passRate%"
# Show recent failures
Write-Host "`nRecent issues:"
$log | Where-Object { $.DimensionsPass -eq "FAIL" } | Select-Object  →
    ↪ ProjectName, Notes
```

Accessibility-Focused QA Best Practices

Why Measurement-Based Testing Matters Traditional visual QA (looking at surface finish, checking dimensions by eye) is inherently inaccessible. This appendix prioritizes:

1. Caliper measurements: Objective, quantifiable
2. Weight verification: Numerical result
3. Functional testing: Pass/fail criteria
4. Go/No-Go gauges: Tactile pass/fail
5. Test jigs: Can be shared/standardized

Screen Reader Integration When documenting QA results:

```
[YES] GOOD: "Bracket width measured 49.8mm (spec 50+/-1mm)"
[NO] AVOID: "Bracket looks good" (not measurable)
[YES] GOOD: "Part weighs 298g (spec 300+/-10g)"
[NO] AVOID: "Feels about right" (subjective)
[YES] GOOD: "Slides through go-gauge, blocks no-go-gauge"
[NO] AVOID: "Hole looks the right size" (visual only)
```

Documentation Template (Accessible)

```
## Part: Bracelet Holder Peg (Test Date: Jan 15, 2024)

### Dimensional Verification
| Dimension | Specification | Measurement 1 | Measurement 2 | Measurement 3 | Average | Result |
|--------------|----------------|---------------|---------------|---------------|---------|------------|
| Peg Diameter | 6.0 +/- 0.5mm | 5.9mm | 6.0mm | 5.8mm | 5.9mm | [YES] PASS |
| Peg Length | 25.0 +/- 0.5mm | 25.1mm | 25.0mm | 25.0mm | 25.03mm | [YES] PASS |

### Functional Test
- [x] Slides into bracelet loop easily (no forcing)
- [x] No cracks visible or felt
- [x] Survives 1-hour load test with 20 bracelets
- [x] No permanent deformation after load removed

### Overall Result
[YES] PASS - All dimensions within tolerance; functionally sound
```

Summary

Key Principles:

1. Plan before printing: Define tolerances and test methods
2. Measure precisely: Use calibrated tools (digital calipers, scale)
3. Test functionally: Will parts assemble and work?
4. Document quantitatively: Use numbers, not adjectives
5. Iterate based on data: If test fails, adjust design or process

When to Use Each Test:

Situation	Recommended Test
Prototype/mockup	Linear dimensions only
Assembly with other parts	Tolerance stack-up analysis
Load-bearing part	Functional load test
Repeated-use item	Durability cycling test
Multi-part design	Assembly fit test

Accessibility Reminder:

All QA testing in this appendix is measurement-based and non-visual, making it equally accessible to all users. The goal is objective, quantifiable verification-not subjective assessment.

Appendix D: Advanced OpenSCAD Concepts

This appendix covers specialized topics for experienced users seeking to tackle complex parametric designs. These are optional topics not required for foundational mastery but valuable for professional application development.

Topic 1: Gears and Mechanical Components

Gears are one of the most challenging parametric designs, requiring careful calculation of tooth geometry. Understanding gear mathematics enables creation of mechanical systems-power transmission, speed reduction, or precise positioning.

Gear Terminology

- Pitch Diameter (PD): The reference diameter for meshing calculations
- Module: PD divided by tooth count; determines tooth size
- Pressure Angle: Typically 20deg or 14.5deg; affects tooth shape and strength
- Clearance: Space between teeth to allow smooth meshing
- Backlash: Intentional gap to prevent binding at tolerance extremes

Simple Involute Gear Algorithm

```
// Simplified gear with circular teeth (adequate for most 3D prints)
function gear_module(pitch_diameter, teeth_count) = pitch_diameter /  →
     ↪ teeth_count;
module simple_gear(pitch_diameter, teeth_count, bore_diameter, thickness) {
  module_m = gear_module(pitch_diameter, teeth_count);
  outer_diameter = pitch_diameter + 2 * module_m;
  tooth_angle = 360 / teeth_count;
  difference() {
    // Main gear body
    cylinder(r=outer_diameter/2, h=thickness, $fn=teeth_count*4);
    // Center bore
    cylinder(r=bore_diameter/2, h=thickness + 2);
  }
  // Add teeth as small protrusions
  for (i = [0:teeth_count-1]) {
    angle = i * tooth_angle;
    translate([
      (pitch_diameter/2 + module_m/2) * cos(angle),
      (pitch_diameter/2 + module_m/2) * sin(angle),
      0
    ])
      rotate([0, 0, angle])
        cube([module_m * 0.8, module_m, thickness], center=true);
  }
}
// Create meshing gear pair
module gear_pair_demo() {
  // Gear 1: 20 teeth, 40mm pitch diameter
  simple_gear(40, 20, 8, 10);
  // Gear 2: 30 teeth, positioned to mesh
  center_distance = (40 + 60) / 4; // Sum of radii
  translate([center_distance, 0, 0])
    simple_gear(60, 30, 8, 10);
}
gear_pair_demo();
```

Practical Application: Servo Gearbox

```
// Parametric servo gearbox with 3:1 reduction
module servo_gearbox(motor_torque, reduction_ratio, bore_size) {
  // Input gear (motor)
  input_teeth = 12;
  input_pd = 30;
  // Output gear (load)
  output_teeth = input_teeth * reduction_ratio;
  output_pd = input_pd * reduction_ratio;
  // Gearbox housing
  housing_size = output_pd + 40;
  difference() {
    // Main box
    cube([housing_size, housing_size, 30]);
    // Interior chamber
    translate([20, 20, 5])
      cube([housing_size - 40, housing_size - 40, 25]);
  }
  // Mount input gear
  translate([housing_size/4, housing_size/2, 15]) {
    simple_gear(input_pd, input_teeth, bore_size, 15);
    // Motor coupling
    cylinder(r=bore_size/2, h=5);
  }
  // Mount output gear
  translate([3*housing_size/4, housing_size/2, 15])
    simple_gear(output_pd, output_teeth, bore_size, 15);
}
servo_gearbox(10, 3, 5);
```

Belt and Pulley Systems

```
// Timing pulley with tooth grooves
module timing_pulley(bore_diameter, pitch_diameter, teeth_count, width) {
  module_m = pitch_diameter / teeth_count;
  tooth_height = module_m * 0.3;
  difference() {
    // Main pulley body
    cylinder(r=pitch_diameter/2 + tooth_height/2, h=width);
    // Center bore
    cylinder(r=bore_diameter/2, h=width + 2);
    // Tooth grooves
    for (i = [0:teeth_count-1]) {
      angle = i * (360 / teeth_count);
      rotate([0, 0, angle])
        translate([pitch_diameter/2, -module_m*0.3, 0])
          cube([module_m * 0.6, module_m * 0.6, width], center=true);
    }
  }
}
// Belt drive demonstration
module belt_drive_system() {
  // Motor pulley (input)
  timing_pulley(5, 20, 16, 10);
  // Load pulley (output) - positioned for belt routing
  translate([80, 0, 0])
    timing_pulley(5, 40, 32, 10);
}
belt_drive_system();
```

Topic 2: Batch Processing and Statistical Analysis

Professional applications often generate many variants and need to analyze results. Batch processing enables systematic exploration of parameter spaces.

Parameter Sweep Generation

```
// Generate models with all combinations of parameters
// Define parameter ranges
wall_thicknesses = [1, 1.5, 2, 2.5, 3];
part_sizes = [20, 30, 40, 50];
materials = ["pla", "petg", "nylon"];
// Theory: Generate 5 x 4 x 3 = 60 variants automatically
// In practice, export each to separate STL for analysis
```

Statistical Summary Generation Script

```
@echo off
rem analyze_batch.bat - Analyze batch results (CMD / Batch)
setlocal enableextensions enabledelayedexpansion
set "OUTPUT_DIR=batch_results"
set "REPORT_FILE=batch_analysis.csv"
echo Part,Wall,Size,File_Size_KB,Est_Print_Time_min,Est_Weight_g > "%REPORT_FILE%"

for %%w in (1 2 3) do (
  for %%s in (20 30 40 50) do (
    set "SCAD_FILE=part_%%s_wall%%w.scad"
    set "STL_FILE=%OUTPUT_DIR%\%%~nSCAD_FILE.stl"
    "C:\Program Files\OpenSCAD\openscad.exe" -D "part_size=%%s" -D "wall=%%w" -o "%OUTPUT_DIR%\part_%%s_wall%%w.stl" src\main.scad
    if exist "%OUTPUT_DIR%\part_%%s_wall%%w.stl" (
      for /f "usebackq" %%F in (`powershell -NoProfile -Command "(Get-Item -Path '%OUTPUT_DIR%\part_%%s_wall%%w.stl').Length"`) do set FILE_BYTES=%%F
      set /a FILE_KB=FILE_BYTES / 1024
      set /a EST_TIME=(FILE_KB / 1024) * 60
      set /a EST_WEIGHT=FILE_KB / 1024
    ) else (
      set FILE_KB=0
      set EST_TIME=0
      set EST_WEIGHT=0
    )
    echo part_%%s_wall%%w,%%w,%%s,!FILE_KB!,!EST_TIME!,!EST_WEIGHT!>>"%REPORT_FILE%"
  )
)
endlocal
echo Analysis complete. Results in %REPORT_FILE%
```

```
# analyze_batch.ps1 - Analyze batch results (PowerShell)
$OutputDir = "batch_results"
$ReportFile = "batch_analysis.csv"
"Part,Wall,Size,File_Size_KB,Est_Print_Time_min,Est_Weight_g" | Out-File -FilePath $ReportFile -Encoding utf8

foreach ($wall in @(1.0,1.5,2.0,2.5,3.0)) {
  foreach ($size in @(20,30,40,50)) {
    $scad = "part_${size}_wall${wall}.scad"
    $stl = Join-Path $OutputDir ("{0}.stl" -f ([System.IO.Path]::GetFileNameWithoutExtension($scad)))
    & "C:\Program Files\OpenSCAD\openscad.exe" -D ("part_size={0}" -f $size) -D ("wall={0}" -f $wall) -o $stl src\main.scad
    if (Test-Path $stl) {
      $bytes = (Get-Item $stl).Length
      $kb = [math]::Round($bytes / 1KB, 2)
      $est_time = [math]::Round(($kb / 1024) * 60, 2)
      $est_weight = [math]::Round($kb / 1024, 2)
    } else {
      $kb = 0; $est_time = 0; $est_weight = 0
    }
    "{0},{1},{2},{3},{4},{5}" -f ("part_{0}_wall{1}" -f $size,$wall), $wall, $size, $kb, $est_time, $est_weight | Out-File -FilePath $ReportFile -Append -Encoding utf8
  }
}
Write-Host "Analysis complete. Results in $ReportFile"
```

```
#!/bin/bash
# analyze_batch.sh - Analyze batch results statistically
```

```bash
OUTPUT_DIR="batch_results"
REPORT_FILE="batch_analysis.csv"

echo "Part,Wall,Size,File_Size_KB,Est_Print_Time_min,Est_Weight_g" > "$REPORT_FILE"

for wall in 1.0 1.5 2.0 2.5 3.0; do
  for size in 20 30 40 50; do
    # Generate model
    SCAD_FILE="part_${size}_wall${wall}.scad"
    STL_FILE="$OUTPUT_DIR/${SCAD_FILE%.scad}.stl"

    # Export to STL
    openscad -D "part_size=$size" -D "wall=$wall" \
      -o "$STL_FILE" src/main.scad

    # Get file size
    FILE_SIZE=$(stat -c%s "$STL_FILE" 2>/dev/null | awk '{print $1/1024}')

    # Estimate print time (rough: 1 hour per MB)
    EST_TIME=$((FILE_SIZE / 1024 * 60))

    # Estimate weight (rough: 1g per MB for PLA)
    EST_WEIGHT=$((FILE_SIZE / 1024))

    # Log results
    echo "part_${size}_wall${wall},$wall,$size,$FILE_SIZE,$EST_TIME,$EST_WEIGHT" >> "$REPORT_FILE"
  done
done

echo "Analysis complete. Results in $REPORT_FILE"
```

Data-Driven Design Selection

```openscad
// Load results and select optimal configuration
// wall=2.0mm gives best balance of strength/weight for most sizes
function optimal_configuration(target_size, priority) =
  priority == "strength" ? 3.0 :
  priority == "speed" ? 1.0 :
  priority == "balanced" ? 2.0 :
  2.0; // Default
module batch_optimized_part(size, priority) {
  wall = optimal_configuration(size, priority);
  echo(str("Generating optimized part: size=", size, " wall=", wall, " priority=", priority));
  // Create part with optimal parameters
  difference() {
    cube([size, size, size]);
    translate([wall, wall, wall])
      cube([size - 2*wall, size - 2*wall, size]);
  }
}
// Generate three variants with different priorities
batch_optimized_part(40, "strength"); // thickest walls
translate([60, 0, 0]) batch_optimized_part(40, "speed"); // thinnest walls
translate([120, 0, 0]) batch_optimized_part(40, "balanced"); // medium walls
```

Topic 3: Performance Optimization

Complex models can take hours to render. Strategic optimization keeps iteration cycles fast.

Measuring Render Performance

```bash
# Time how long rendering takes
time openscad -o output.stl -D "quality=32" src/main.scad
```

```
# Output: real 5m 32s, user 5m 28s, sys 0m 2s
```

Resolution Parameter Strategy

```
// Use lower resolution during development, high during export
quality = $preview ? 16 : 64; // 16 segments in preview, 64 in export
module efficient_sphere(radius) {
  sphere(r=radius, $fn=quality);
}
// In array operations, efficiency matters most
module circular_array(radius, count) {
  for (i = [0:count-1]) {
    angle = (360 / count) * i;
    x = radius * cos(angle);
    y = radius * sin(angle);
    translate([x, y, 0])
      efficient_sphere(5); // Uses quality parameter
  }
}
circular_array(30, 100); // 100 spheres: slow with quality=64!
```

Cache Complex Calculations

```
// Pre-compute expensive calculations outside loops
// SLOW: Recalculates sin/cos for every iteration
module slow_spiral() {
  for (i = [0:200]) {
    angle = i * 2;
    x = 40 * cos(angle);
    y = 40 * sin(angle);
    z = i * 0.5;
    translate([x, y, z])
      sphere(r=1);
  }
}
// FAST: Pre-compute positions, then use them
spiral_positions = [for (i = [0:200]) [
  40 * cos(i * 2),
  40 * sin(i * 2),
  i * 0.5
]];
module fast_spiral() {
  for (pos = spiral_positions)
    translate(pos)
      sphere(r=1);
}
// Use the fast version
fast_spiral();
```

Simplification During Preview

```
// Complex model that simplifies in preview mode
module detailed_model() {
  if ($preview) {
    // Simplified version for fast preview
    cube([100, 100, 50]);
  } else {
    // Detailed version for export
    difference() {
      cube([100, 100, 50]);
      // Complex inner geometry
      for (x = [10:10:90])
        for (y = [10:10:90])
          translate([x, y, 20])
            cylinder(h=15, d=3, $fn=32);
    }
  }
}
detailed_model();
```

Profile-Driven Optimization

```bash
#!/bin/bash
# profile_render.sh - Identify slow rendering hotspots

echo "Profiling render performance..."

for fn_value in 8 16 32 64 128; do
  echo ""
  echo "Testing with \$fn=$fn_value..."

  /usr/bin/time -v openscad -D "fn=$fn_value" \
    -o test_$fn_value.stl src/main.scad 2>&1 | \
    grep "Maximum resident set size"
done
```

Topic 4: Print Orientation and Support Structure Algorithms

Strategic orientation dramatically affects support material, print time, and surface quality.

Strength Orientation Analysis

```
// Different orientations affect strength differently
// SCENARIO: Bracket with cantilever load
module bracket() {
  difference() {
    cube([50, 100, 10]);
    translate([25, 50, 5]) cylinder(r=5, h=10);
  }
}
// Orientation A: Lay flat (XY plane) - WEAK for cantilever
echo("Orientation A: Flat - Low strength for cantilever");
bracket();
// Orientation B: Stand tall (Z axis) - STRONG for cantilever
translate([70, 0, 0]) {
  echo("Orientation B: Tall - High strength for cantilever");
  rotate([90, 0, 0]) bracket();
}
// Orientation C: Diagonal - MODERATE strength
translate([0, 150, 0]) {
  echo("Orientation C: Diagonal - Moderate strength");
  rotate([45, 0, 0]) bracket();
}
```

Support Material Minimization

```
// Design features to reduce supports needed
// POOR: Overhanging feature needs extensive supports
module poor_design() {
  cube([50, 50, 50]);
  translate([25, 25, 50])
    cube([30, 30, 10]); // 30mm overhang!
}
// GOOD: Bridge angle keeps overhang under 45deg
module good_design() {
  cube([50, 50, 50]);
  // Add ramp instead of overhang
  translate([20, 20, 50])
    rotate([20, 0, 0])
      cube([30, 30, 10]);
}
```

```
// Compare
poor_design();
translate([80, 0, 0])
  good_design();
```

Bridge Span Calculation

```
// Calculate maximum unsupported span for given material/settings
function max_bridge_span(material) =
  material == "pla" ? 10 :
  material == "petg" ? 12 :
  material == "nylon" ? 15 :
  material == "abs" ? 8 :
  10; // Default
module bridged_connector(material, gap_width) {
  max_span = max_bridge_span(material);
  if (gap_width <= max_span) {
    echo(str("Bridge OK: gap ", gap_width, "mm fits within ", max_span, "mm  →
        ↪ limit"));
    // Create part with bridge
    union() {
      cube([50, 10, 10]); // Side A
      translate([gap_width, 0, 0])
        cube([50, 10, 10]); // Side B
      // Bridge connecting them
      translate([0, 5, 10])
        cube([gap_width + 100, 2, 1]);
    }
  } else {
    echo(str("ERROR: gap ", gap_width, "mm exceeds ", max_span, "mm max  →
        ↪ bridge span"));
  }
}
bridged_connector("pla", 8); // OK: 8 < 10
// bridged_connector("pla", 15); // ERROR: 15 > 10
```

Slicing Parameter Calculation

```
// Calculate optimal slicing parameters based on geometry
function recommended_layer_height(nozzle_diameter, quality_priority) =
  quality_priority == "fast" ? nozzle_diameter * 0.75 :
  quality_priority == "balanced" ? nozzle_diameter * 0.5 :
  quality_priority == "detailed" ? nozzle_diameter * 0.25 :
  nozzle_diameter * 0.5;
function recommended_infill(structural_load, safety_factor) =
  structural_load < 1 ? 10 : // Decorative: minimal fill
  structural_load < 5 ? 20 : // Light duty: low fill
  structural_load < 20 ? 50 : // Medium duty: standard fill
  100; // High duty: solid
module optimized_part(
  nozzle_0p4,
  quality_priority,
  structural_load,
  safety_factor
) {
  layer_h = recommended_layer_height(nozzle_0p4, quality_priority);
  infill_pct = recommended_infill(structural_load, safety_factor);
  echo(str("Recommended layer height: ", layer_h, "mm"));
  echo(str("Recommended infill: ", infill_pct, "%"));
  // Create part
  cube([50, 50, 30]);
}
optimized_part(0.4, "detailed", 15, 2);
```

Topic 5: Recursive Function Patterns

Recursion enables elegant solutions to problems with self-similar structure: trees, fractals, nested components.

Basic Recursive Pattern

```
// Tree structure: branches recursively smaller
function tree_depth(level) = level > 0 ? level + tree_depth(level - 1) : 0;
// Calculate: tree_depth(5) = 5 + 4 + 3 + 2 + 1 = 15
result = tree_depth(5);
echo(result); // Prints: 15
// Recursive tree drawing
module tree_branch(length, angle, recursion_depth) {
  if (recursion_depth > 0) {
    // Draw this branch
    rotate([angle, 0, 0])
      cube([2, 2, length]);
    // Recursively draw sub-branches
    translate([0, 0, length])
      rotate([-angle/2, 0, 0])
        tree_branch(length * 0.7, angle, recursion_depth - 1);
    translate([0, 0, length])
      rotate([angle/2, 0, 0])
        tree_branch(length * 0.7, angle, recursion_depth - 1);
  }
}
tree_branch(30, 25, 4); // Tree with 4 levels of recursion
```

Fractal Generation

```
// Sierpinski Triangle fractal
function sierpinski_triangle(size, depth) =
  depth == 0 ?
    [[0, 0], [size, 0], [size/2, size * sqrt(3)/2]] :
    // Recursive case: three smaller triangles at corners
    concat(
      sierpinski_triangle(size/2, depth - 1),
      sierpinski_triangle(size/2, depth - 1),
      sierpinski_triangle(size/2, depth - 1)
    );
// Create 3D fractal structure
module fractal_spiral(base_size, depth, height_per_level) {
  if (depth > 0) {
    // Create current level
    cube([base_size, base_size, 5], center=true);
    // Recursively create smaller level on top
    translate([0, 0, height_per_level])
      scale([0.6, 0.6, 1])
        fractal_spiral(base_size, depth - 1, height_per_level);
  }
}
fractal_spiral(40, 5, 10); // 5-level spiral
```

Nested Component Assembly

```
// Russian doll boxes: each box contains a smaller copy
module russian_doll(size, wall, nesting_depth) {
  if (nesting_depth > 0) {
    // Current level: hollow box
    difference() {
      cube([size, size, size]);
      translate([wall, wall, wall])
        cube([size - 2*wall, size - 2*wall, size - wall]);
    }
    // Recursively place next smaller doll inside
    interior_size = size - 4*wall;
    translate([2*wall, 2*wall, wall])
      russian_doll(interior_size, wall, nesting_depth - 1);
  } else {
    // Smallest doll (solid)
    cube([size, size, size]);
  }
}
russian_doll(60, 2, 4); // 4 nested boxes
```

Performance Considerations

```
// Recursion can be expensive - monitor depth
// EFFICIENT: Tail recursion (result computed immediately)
function sum_to(n, accumulator) =
  n == 0 ? accumulator :
  sum_to(n - 1, accumulator + n);
result = sum_to(100, 0); // Computes: 5050
// INEFFICIENT: Deep recursion without optimization
function fibonacci(n) =
  n <= 1 ? n :
  fibonacci(n-1) + fibonacci(n-2);
// fibonacci(20) recalculates sub-problems thousands of times
// Don't use for depth > 20 without memoization
// BETTER: Use iteration where possible
function fibonacci_iter(n) =
  let(
    fib = [for (i = [0:n])
      i <= 1 ? i :
      let(prev = [for (j = [0:i-1]) i==j ? 1 : i==j+1 ? 0 : 0])
        1 // Placeholder
    ]
  ) fib[n];
```

Practical Example: Cable Management

```
// Recursive cable tray with branching paths
module cable_tray(width, depth, height, levels, branch_angle) {
  if (levels > 0) {
    // Main tray section
    difference() {
      cube([width, depth, height]);
      translate([2, 2, 2])
        cube([width - 4, depth - 4, height - 2]);
    }
    // Left branch (recursively smaller)
    translate([-width/2, 0, height])
      rotate([0, 0, -branch_angle])
        cable_tray(width * 0.6, depth, height, levels - 1, branch_angle);
    // Right branch
    translate([width/2, 0, height])
      rotate([0, 0, branch_angle])
        cable_tray(width * 0.6, depth, height, levels - 1, branch_angle);
  }
}
cable_tray(40, 10, 5, 3, 30); // 3-level branching cable tray
```

Summary: When to Use Each Advanced Technique

Technique	Use Case	Complexity	Performance
Gears	Mechanical power transmission	High	Medium

Technique	Use Case	Complexity	Performance
Batch Processing	Design space exploration	Medium	Depends on scope
Performance Optimization	Reducing render time	Medium	High payoff
Orientation Analysis	Strength optimization	Medium	Simulation cost
Recursion	Fractal/nested structures	High	Can be expensive

References and Further Learning

- OpenSCAD Documentation - Advanced Features: https://en.wikibooks.org/wiki/OpenSCAD_User_Manual
- Gear Design Theory: https://www.mekanizmalar.com/
- 3D Print Orientation Optimization: https://stratasys.com/
- Fractal Geometry: https://en.wikipedia.org/wiki/Fractal
- Recursive Algorithms: https://www.khanacademy.org/computing/computer-science/algorithms

For Educators and Students

These advanced topics are intended for:

- Experienced users tackling professional applications
- Specialized problem domains (robotics, mechanical design, etc.)
- Research and optimization work
- Custom library and framework development

For accessibility:

- All recursive examples include base cases clearly marked
- Each section provides simplified versions before advanced variants
- Code comments explain both ”what” and ”why”
- Practical examples show real-world applications
- Performance considerations documented to prevent frustration

Glossary — Organized by Topic

This glossary groups and alphabetizes terms by topic. It includes core 3D design and printing vocabulary, OpenSCAD-specific terms, command-line and coding terms, and accessibility/tooling references.

General

- Additive manufacturing: Processes that build objects by adding material layer by layer (3D printing).
- CAD (Computer-Aided Design): Software for creating 2D/3D models; OpenSCAD is a CAD tool focused on programmatic modeling.
- Design: An OpenSCAD program or other model created for fabrication or simulation.
- Parameter / parametric design: Designing with variables so a model can be changed by altering inputs.
- Preview: Fast, approximate model view (OpenSCAD `F5`).
- Render: Full evaluation of geometry (OpenSCAD `F6`) used before export.

3D Printing

- Bed adhesion: Methods to keep the first layer attached to the build plate (tape, glue, heated bed).
- Brim: A single-layer flat ring around a model to improve adhesion.
- Filament: Thermoplastic material (PLA, PETG, ABS, TPU) used by FDM printers.
- G-code: The machine instructions that drive a printer (moves, extrusion, temps).
- Infill: Internal lattice/pattern inside a print, expressed as a percentage.
- Layer height: Vertical thickness of each printed layer; impacts quality and time.
- Nozzle / Hot end: The heated tip that melts and extrudes filament.
- Overhang: Model area with little/no support underneath; often needs support structures.
- Slicer: Software that converts 3D models (STL) into G-code for a specific printer.
- Support structures: Temporary scaffolding printed to support overhangs.
- STL: Common 3D model file format for printing (triangle mesh).
- Warping: Distortion from uneven cooling that lifts part edges.

OpenSCAD — Language & Modeling

- CSG (Constructive Solid Geometry): Building complex shapes by combining primitives with boolean ops.
- Children / children(): The nested block passed into a module; accessible with `children()`.
- Linear extrusion: `linear_extrude()` — creates 3D geometry from a 2D profile.
- Module: A named reusable block of OpenSCAD code defined with `module name() {}`.
- Parameter: Named values used to control geometry and behavior inside modules/functions.

- Primitive: Built-in shapes such as `cube()`, `cylinder()`, `sphere()`, `polyhedron()`.
- Preview vs Render: Preview (`F5`) is fast; Render (`F6`) evaluates final geometry.
- rotate() / translate() / scale(): Common transforms to orient and place geometry.
- union() / difference() / intersection(): Boolean operations for combining or subtracting solids.
- $fn: Special variable controlling polygonal approximation (number of faces) for rounded primitives.

3dMake / Build Automation (CLI-based toolchain)

- Artifact: Any generated file from the build pipeline (STL, G-code, logs).
- Automation / Headless render: Running builds or OpenSCAD without a GUI, typically via scripts or CI.
- Build script: Script that automates rendering, exporting, slicing, and packaging.
- Parameter file: External JSON/TOML providing parameters for reproducible builds.
- Pipeline: Ordered steps from source to final artifact (render ⌧ export ⌧ slice ⌧ G-code).

PowerShell & Automation

- Cmdlet: Lightweight PowerShell command (e.g., `Get-ChildItem`).
- Execution policy: Controls whether PowerShell scripts are allowed to run on the machine.
- Object: PowerShell passes rich objects (not text) between cmdlets.
- Pipeline (PowerShell): Passing output of one cmdlet as input to another using `|`.
- Remote session: Running PowerShell commands on a remote host via `Enter-PSSession` or `Invoke-Command`.

Command-Line & Shell

- Argument / Flag: Values or options passed to a command (e.g., `-r`, `--verbose`).
- Bash: A common Unix shell used interactively and in scripts; many curriculum examples reference Git Bash on Windows.
- CLI (Command-Line Interface): Text-based interface for interacting with programs via commands.
- Environment variable: Named values available to processes (e.g., `PATH`, `HOME`).

- Exit code: Numeric status returned by a command (`0` = success conventionally).
- Pipe (`|`): Send stdout of one command to stdin of another.
- Redirection: > and >> to write stdout to files; `2>` for stderr.
- Shell script: A file of shell commands (Bash: `.sh`) executed by a shell.
- Standard streams: `stdin`, `stdout`, `stderr` for input, normal output, and error output.
- SSH / SCP: Secure remote shell/file copy tools for remote management and transfer.
- grep / sed / awk / jq: Common command-line text processing and JSON tools.

Coding & Programming Terms

- API (Application Programming Interface): A defined interface for interacting with software or web services.
- Array / List: Ordered collection of values.
- Commit / Branch / Repository (git): Version control concepts: commits are snapshots, branches hold parallel work, repo stores history.
- CI (Continuous Integration): Automated builds and tests triggered by changes to source code.
- Function: Named reusable block of code that returns a value or performs an action.
- JSON / TOML / YAML: Common structured data formats used for configuration and parameter files.
- Library / Package / Module: Reusable code collections that your code can import.
- Linting: Static analysis to detect style issues or simple bugs in code.
- Loop / Conditional: Control flow constructs (for, while, if/else) used to repeat or branch logic.
- Regex (Regular expression): Pattern syntax for searching and transforming text.
- Unit test / Integration test: Automated tests that validate small units or integrated components.
- Virtual environment: Isolated Python environment (venv, virtualenv) to manage dependencies.

Tools & Resources

- 3DMake: CLI workflow tool referenced in the curriculum for headless builds and automation.
- Git / GitHub: Version control and hosting used for code, examples, and CodeSolutions.
- Notable docs: NVDA, OpenSCAD, PrusaSlicer, and vendor sites are referenced throughout—links are in the Further Reading section.

Accessibility (a11y)

- Screen reader: Software that reads text aloud (NVDA, JAWS) or provides braille output.
- Braille display: Hardware that renders text as tactile braille cells.
- Keyboard navigation: Ensuring tools and web content are usable without a mouse.

Further Reading

- http://github.com/tdeck/3dmake
- https://all3dp.com/1/3d-printer-filament-types-3d-printing-3d-filament/
- https://bambulab.com/en/download
- https://bambulab.com/en/download/studio
- https://code.visualstudio.com/
- https://code.visualstudio.com/docs/editor/accessibility
- https://community.bambulab.com/
- https://community.prusa3d.com/
- https://creativecommons.org/licenses/by-nc-nd/4.0/
- https://discord.gg/F2Nx2VxTB7
- https://docs.cloud.google.com/vertex-ai/generative-ai/docs/start/get-started-with-gemini-3
- https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/cd
- https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/dir
- https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands-glossary
- https://docs.microsoft.com/powershell
- https://docs.microsoft.com/powershell/
- https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about_error_handling
- https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about_foreach
- https://docs.microsoft.com/powershell/module/microsoft.powershell.core/about/about_functions_advanced
- https://docs.microsoft.com/powershell/scripting/
- https://docs.microsoft.com/windows-server/administration/windows-commands/windows-commands
- https://docs.microsoft.com/windows/terminal/
- https://docs.prusa3d.com/en/
- https://docs.prusa3d.com/en/guide/39012-validation-tools/
- https://docsbot.ai/prompts/technical/openscad-prompt-creation
- https://doh.wa.gov/community-and-environment/schools/3d-printers
- https://ehs.osu.edu/kb/3d-printer-safety

- https://ehs.utk.edu/index.php/table-of-policies-plans-procedures-guides/3d-printer-safety/
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/FAQ
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Modules
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Variables
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations#minkowski
- https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Using_the_2D_Subsystem
- https://en.wikibooks.org/wiki/OpenSCADUserManual
- https://en.wikipedia.org/wiki/Fractal
- https://example.com
- https://forum.openscad.org/
- https://git-scm.com
- https://git-scm.com/
- https://git-scm.com/book/
- https://git-scm.com/book/en/v2
- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository
- https://git-scm.com/download/win
- https://gitforwindows.org/
- https://github.com/BelfrySCAD/BOSL2
- https://github.com/ProgrammingWithOpenSCAD/CodeSolutions
- https://github.com/ProgrammingWithOpenSCAD/CodeSolutions/tree/main/1_3D-Shapes
- https://github.com/ProgrammingWithOpenSCAD/CodeSolutions/tree/main/5_Modules
- https://github.com/ProgrammingWithOpenSCAD/CodeSolutions/tree/main/AppendixB_OpenSCAD-Visual-Reference
- https://github.com/revarbat/BOSL/wiki
- https://github.com/revarbat/BOSL2
- https://github.com/SoftFever/OrcaSlicer
- https://github.com/SoftFever/OrcaSlicer/releases
- https://github.com/SoftFever/OrcaSlicer/wiki
- https://github.com/supermerill/SuperSlicer
- https://github.com/tdeck/3dmake
- https://github.com/tdeck/3dmake/blob/main/docs/automation.md
- https://github.com/tdeck/3dmake/blob/main/docs/examples.md
- https://github.com/tdeck/3dmake/blob/main/docs/terminalquickstart.

md
- https://github.com/tdeck/3dmake/blob/main/e2e_test.py
- https://github.com/tdeck/3dmake/releases/latest/download/3dmakelinux.tar.gz
- https://github.com/tdeck/3dmake/releases/latest/download/3dmakewindows.zip
- https://github.com/tdeck/3dmake/wiki/Library-Management
- https://github.com/tdeck/3dmake/wiki/Troubleshooting
- https://help.prusa3d.com
- https://help.prusa3d.com/category/prusaslicer_204
- https://help.prusa3d.com/materials
- https://learn.cadhub.xyz/blog/openscad-review/
- https://learn.microsoft.com/en-us/training/modules/
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cd
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cmd
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/doskey
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/find
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/findstr
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/path
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/redirection
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/set
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/setx
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/start
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/type
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/where
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands-glossary
- https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about_operators#call-operator-
- https://learn.microsoft.com/powershell/module/microsoft.powershell.core/about/about_profiles
- https://learn.microsoft.com/powershell/module/microsoft.powershell.cor

e/get-command

- https://learn.microsoft.com/powershell/module/microsoft.powershell.core/invoke-history
- https://learn.microsoft.com/powershell/module/microsoft.powershell.management/copy-item
- https://learn.microsoft.com/powershell/module/microsoft.powershell.management/get-childitem
- https://learn.microsoft.com/powershell/module/microsoft.powershell.management/get-content
- https://learn.microsoft.com/powershell/module/microsoft.powershell.management/new-item
- https://learn.microsoft.com/powershell/module/microsoft.powershell.management/remove-item
- https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/measure-object
- https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/out-file
- https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/select-string
- https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/set-alias
- https://learn.microsoft.com/powershell/module/psreadline/about/about_psreadline_functions
- https://learn.microsoft.com/powershell/scripting/learn/shell/manipulating-items
- https://learn.microsoft.com/powershell/scripting/learn/shell/navigate-the-filesystem
- https://learn.microsoft.com/powershell/scripting/learn/shell/using-environment-variables
- https://learn.microsoft.com/powershell/scripting/learn/shell/using-history
- https://learn.microsoft.com/powershell/scripting/learn/shell/using-the-pipeline
- https://learn.microsoft.com/powershell/scripting/learn/shell/using-wildcards
- https://learn.microsoft.com/powershell/scripting/overview
- https://learn.microsoft.com/powershell/scripting/windows-powershell/ise/accessibility-in-windows-powershell-ise
- https://lib.utah.edu/protospace.php
- https://linux.die.net/man/1/which
- https://linuxcommand.org/
- https://linuxcommand.org/lc3lts0020.php
- https://linuxcommand.org/lc3lts0050.php
- https://linuxize.com/post/how-to-add-directory-to-path-in-linux/
- https://linuxize.com/post/linux-cd-command/
- https://makesaltlake.org/

- https://mrhunsaker.github.io/VI_3DMake_OpenSCAD_Curriculum/
- https://news.ycombinator.com/item?id=46338565
- https://nostarch.com/programmingopenscad
- https://notepad-plus-plus.org/
- https://notepad-plus-plus.org/downloads/
- https://notepad-plus-plus.org/online-help/
- https://openscad.org
- https://openscad.org/cheatsheet/
- https://openscad.org/documentation.html
- https://openscad.org/downloads.html
- https://poshcode.gitbook.io/powershell-faq/src/getting-started/environment-variables
- https://poshcode.gitbook.io/powershell-faq/src/getting-started/file-management
- https://poshcode.gitbook.io/powershell-faq/src/getting-started/filesystem-navigation
- https://programmingwithopenscad.github.io/
- https://programmingwithopenscad.github.io/learning.html
- https://programmingwithopenscad.github.io/quick-reference.html
- https://rust-lang.github.io/mdBook/
- https://services.slcpl.org/creativelab
- https://stratasys.com/
- https://superuser.com/questions/167461/what-are-path-separators-on-windows
- https://support.freedomscientific.com/Content/Documents/Manuals/JAWS/JAWS-Screen-Reader-Help.pdf
- https://support.microsoft.com/en-us/help/22798/windows-11-narrator-get-started
- https://support.microsoft.com/narrator
- https://theprusaslicer.net/how-to-use-prusaslicer/
- https://thomaspietrzak.com/bibliography/gonzalez24.pdf
- https://tldp.org/LDP/Bash-Beginners-Guide/html/
- https://ultimaker.com/software/ultimaker-cura
- https://wiki.bambulab.com
- https://www.a11y-101.com/design/inclusive-documentation
- https://www.aiha.org/
- https://www.all3dp.com/guides/
- https://www.anycubic.com/
- https://www.apache.org/licenses/LICENSE-2.0
- https://www.autodesk.com/products/fusion-360
- https://www.cdc.gov/niosh/blogs/2024/safe-3d-printing.html
- https://www.classcentral.com/subject/openscad
- https://www.freedomscientific.com/
- https://www.freedomscientific.com/products/software/jaws/
- https://www.freedomscientific.com/support/
- https://www.gnu.org/software/bash/manual/

- https://www.gnu.org/software/bash/manual/bash.html
- https://www.gnu.org/software/bash/manual/bash.html#Basic-Shell-Features
- https://www.gnu.org/software/bash/manual/htmlnode/Aliases.html
- https://www.gnu.org/software/bash/manual/htmlnode/Bash-History-Builtins.html
- https://www.gnu.org/software/bash/manual/htmlnode/Bash-History-Facilities.html
- https://www.gnu.org/software/bash/manual/htmlnode/Bash-Startup-Files.html
- https://www.gnu.org/software/bash/manual/htmlnode/Bourne-Shell-Builtins.html
- https://www.gnu.org/software/bash/manual/htmlnode/Commands-For-Completion.html
- https://www.gnu.org/software/bash/manual/htmlnode/Controlling-the-Prompt.html
- https://www.gnu.org/software/bash/manual/htmlnode/Environment.html
- https://www.gnu.org/software/bash/manual/htmlnode/Executing-Scripts.html
- https://www.gnu.org/software/bash/manual/htmlnode/Pattern-Matching.html
- https://www.gnu.org/software/bash/manual/htmlnode/Pipelines.html
- https://www.gnu.org/software/bash/manual/htmlnode/Redirections.html
- https://www.gnu.org/software/coreutils/manual/
- https://www.gnu.org/software/coreutils/manual/html_node/cat-invocation.html
- https://www.gnu.org/software/coreutils/manual/html_node/cp-invocation.html
- https://www.gnu.org/software/coreutils/manual/html_node/ls-invocation.html
- https://www.gnu.org/software/coreutils/manual/html_node/mv-invocation.html
- https://www.gnu.org/software/coreutils/manual/html_node/rm-invocation.html
- https://www.gnu.org/software/coreutils/manual/html_node/touch-invocation.html
- https://www.gnu.org/software/coreutils/manual/html_node/wc-invocation.html
- https://www.gnu.org/software/grep/manual/grep.html
- https://www.hubs.com/knowledge-base/what-is-fdm-3d-printing/
- https://www.ideo.com/perspectives/design-thinking
- https://www.khanacademy.org/computing/computer-science/algorithms
- https://www.matterhackers.com/3d-printer-filament-compare
- https://www.mekanizmalar.com/
- https://www.nist.gov/publications
- https://www.nvaccess.org/

- https://www.nvaccess.org/documentation/
- https://www.nvaccess.org/files/nvda/documentation/userGuide.html
- https://www.packtpub.com/en-us/product/simplifying-3d-printing-with-openscad-9781801813174
- https://www.productschool.com/blog/requirements/
- https://www.prusa3d.com/page/prusaslicer424/
- https://www.prusa3d.com/page/prusaslicer_410/
- https://www.prusa3d.com/support/
- https://www.raise3d.com/ideamaker
- https://www.reddit.com/r/3Dprinting/
- https://www.reddit.com/r/openscad/
- https://www.reddit.com/r/openscad/comments/1fxj8xv/why_is_there_so_little_content_and_community/
- https://www.rit.edu/ehs/3-d-printer-safety
- https://www.sculpteo.com/en/blog/2021/06/17/tolerance-3d-printing/
- https://www.slcolibrary.org/what-we-have/create
- https://www.theblackbookofinnovation.com/lassonde-studios
- https://www.youtube.com/@LukesBlab
- https://www.youtube.com/@TeachingTech
- https://www.zenml.io/llmops-database/llm-powered-3d-model-generation-for-3d-printing
- https://yourdolphin.com/supernova/