Secure scripting (SES)

This module focuses on basic scripting using the Bourne-Again SHell (BASH). It focuses on basic constructs and useful commands, such as conditional, looping, and logical constructs, as well as variables and user and system interfaces. Students create basic scripts and have a foundation upon which to learn more advanced bash scripting, or other scripting languages. One of the exercises is to implement a simple file system scanner, so students practice scripting security applications and administrative scripts.

# learning objectives

Primary learning objectives

Students will understand how to approach a problem to solve it using scripting, and they will understand bottom-up programming techniques suitable for use in scripting. Students will demonstrate proficiency in scripting by using elements of the Bourne shell scripting language, and of scripting languages for various Linux tools.

Associated Sub Learning Objectives

NSA/DHS CAE Knowledge Units for Basic Scripting and Programming (BSP)

**Topics**

1. Implement basic security concepts
   1. Permissions, bounds checking, input validation, type checking and parameter validation
2. Be familiar with the concept and basic implementation of regular expressions.
3. Understand basic data structures and algorithms
4. Basic Boolean logic/operations.
   1. AND / OR / XOR / NOT
5. Scripting on both Windows and Linux
   1. Language (e.g. PERL, Python, BASH, JAVA, VB Scripting, Powershell)
6. Properly apply basic programming constructs and concepts including:
   1. Variables and types (int, float, char, etc.)
   2. Strings, arrays, structures
   3. Sequential and parallel execution
   4. Assignments (:=, =, ++, --, etc.)
   5. Decisions and branching (if, if ... else, elseif, switch, case, etc.)
   6. Loops (for, while, repeat, etc.)
   7. Functions, procedures, and calls

**Outcome:** 1. Demonstrate their proficiency in the use of scripting languages to write simple scripts (e.g., to automate system administration tasks).

**Outcome:** 2. Write simple linear and looping scripts.

**Outcome:** 3. Write simple and compound conditions within a programming language or similar environment (e.g., scripts, macros, SQL).

**Outcome:** 4. Demonstrate proficiency in the use of a programming language to solve complex problems in a secure and robust manner.

AP Computer Science Principles Course, Big Ideas 1-5

**LO 1.1.1** Students will be able to apply a creative development process when creating computational artifacts.

**EK 1.1.1A** A creative process in the development of a computational artifact can include, but is not limited to, employing nontraditional, nonprescribed techniques; the use of novel combinations of artifacts, tools and techniques; and the exploration of personal curiosities.

**EK 1.1.1B** Creating computational artifacts employs an iterative and often exploratory process to translate ideas into tangible form.

**LO 1.2.2** Students will be able to create a computational artifact using computing tools and techniques to solve a problem.   
More specifically, this module addresses the following EK statements:

**EK 1.2.2A** Computing tools and techniques can enhance the process of finding a solution to a problem.

**LO 1.2.3** Students will be able to create a new computational artifact by combining or modifying existing artifacts.

**EK 1.2.3A** Creating computational artifacts can be done by combining and modifying existing artifacts or by creating new artifacts.

**EK 1.2.3B** Computation facilitates the creation and modification of computational artifacts with enhanced detail and precision.

**EK 1.2.3C** Combining or modifying existing artifacts can show personal expression of ideas.

**LO 2.1.1** Students will be able to describe the variety of abstractions used to represent data.

More specifically, this module addresses the following EK statements:

**EK 2.1.1A** Digital data is represented by abstractions at different levels.

**EK 2.1.1B** At the lowest level, all digital data are represented by bits.

**EK 2.1.1C** At a higher level, bits are grouped to represent abstractions, including but not limited to numbers, characters, and color.

**LO 2.2.2** Students will be able to use multiple levels of abstraction to write programs.

**EK 2.2.2A** Software is developed using multiple levels of abstractions, such as constants, expressions, statements, procedures, and libraries.

**EK 2.2.2B** Being aware of and using multiple levels of abstractions in developing programs help to more effectively apply available resources and tools to solve problems.

**LO 2.2.3** Students will be able to identify multiple levels of abstractions that are used when writing programs.

More specifically, this module addresses the following EK statements:

**EK 2.2.3B** High-level programming languages provide more abstractions for the programmer and make it easier for people to read and write a program.  
**EK 2.2.3D** In an abstraction hierarchy, higher levels of abstraction (the most general concepts) would be placed toward the top and lower-level abstractions (the more specific concepts) toward the bottom.

**EK 2.2.3J** Applications and systems are designed, developed, and analyzed using levels of hardware, software, and conceptual abstractions.

**EK 2.2.3K** Lower-level abstractions can be combined to make higher-level abstractions, such as short message services (SMS) or email messages, images, audio files, and videos.

**LO 3.3.1** Students will be able to analyze how data representation, storage, security, and transmission of data involve computational manipulation of information.

More specifically, this module addresses the following EK statements:

**EK 3.3.1A** Digital data representations involve trade-offs related to storage, security, and privacy concerns.

**EK 3.3.1G** Data is stored in many formats depending on its characteristics (e.g., size and intended use).

**EK 3.3.1H** The choice of storage media affects both the methods and costs of manipulating the data it contains.

**EK 3.3.1I** Reading data and updating data have different storage requirements.

**LO 4.1.1** Students will be able to develop an algorithm for implementation in a program.

**EK 4.1.1A** Sequencing, selection, and iteration are building blocks of algorithms. **EK 4.1.1B** Sequencing is the application of each step of an algorithm in the order in which the statements are given.

**EK 4.1.1C** Selection uses a Boolean condition to determine which of two parts of an algorithm is used.

**EK 4.1.1D** Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times.

**EK 4.1.1E** Algorithms can be combined to make new algorithms.

**EK 4.1.1F** Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct.

**EK 4.1.1G** Knowledge of standard algorithms can help in constructing new algorithms.

**EK 4.1.1H** Different algorithms can be developed to solve the same problem.

**EK 4.1.1I** Developing a new algorithm to solve a problem can yield insight into the problem.

**LO 4.1.2** Students will be able to express an algorithm in a language.   
More specifically, this module addresses the following EK statements:

**EK 4.1.2A** Languages for algorithms include natural language, pseudocode, and visual and textual programming languages.

**EK 4.1.2B** Natural language and pseudocode describe algorithms so that humans can understand them.

**EK 4.1.2C** Algorithms described in programming languages can be executed on a computer.

**EK 4.1.2E** Some programming languages are designed for specific domains and are better for expressing algorithms in those domains.

**EK 4.1.2F** The language used to express an algorithm can affect characteristics such as clarity or readability but not whether an algorithmic solution exists.

**EK 4.1.2G** Every algorithm can be constructed using only sequencing, selection, and iteration.

**EK 4.1.2I** Clarity and readability are important considerations when expressing an algorithm in a language.

**LO 5.1.2** Students will be able to develop a correct program to solve problems. More specifically, this module addresses the following EK statements:

**EK 5.1.2A** An iterative process of program development helps in developing a correct program to solve problems.

**EK 5.1.2B** Developing correct program components and then combining them helps in creating correct programs.

**EK 5.1.2C** Incrementally adding tested program segments to correct working programs helps create large correct programs.

**EK 5.1.2D** Program documentation helps programmers develop and maintain correct programs to efficiently solve problems.

**EK 5.1.2E** Documentation about program components, such as code segments and procedures, helps in developing and maintaining programs.

**EK 5.1.2F** Documentation helps in developing and maintaining programs when working individually or in collaborative programming environments.

**EK 5.1.2G** Program development includes identifying programmer and user concerns that affect the solution to problems.

**EK 5.1.2I** A programmer’s knowledge and skill affects how a program is developed and how it is used to solve a problem.

**EK 5.1.2J** A programmer designs, implements, tests, debugs, and maintains programs when solving problems.

**LO 5.4.1** Students will be able to evaluate the correctness of a program.

More specifically, this module addresses the following EK statements:

**EK 5.4.1K** Correctness of a program depends on correctness of program components, including code segments and procedures.

**EK 5.4.1L** An explanation of a program helps people understand the functionality and purpose of it.

**EK 5.4.1M** The functionality of a program is often described by how a user interacts with it.

**EK 5.4.1N** The functionality of a program is best described at a high level by what the program does, not at the lower level of how the program statements work to accomplish this.

**LO 5.5.1** Students will be able to employ appropriate mathematical and logical concepts in programming.

More specifically, this module addresses the following EK statements:

**EK 5.5.1A** Numbers and numerical concepts are fundamental to programming.

**EK 5.5.1B** Integers may be constrained in the maximum and minimum values that can be represented in a program because of storage limitations.

EXCLUSION STATEMENT (for EK 5.5.1B): Specific range limitations of all programming languages are beyond the scope of this course and the AP Exam.

**EK 5.5.1D** Mathematical expressions using arithmetic operators are part of most programming languages.

**EK 5.5.1E** Logical concepts and Boolean algebra are fundamental to programming.

**EK 5.5.1F** Compound expressions using *and, or,* and *not* are part of most programming languages.

**EK 5.5.1G** Intuitive and formal reasoning about program components using Boolean concepts helps in developing correct programs.

**EK 5.5.1H** Computational methods may use lists and collections to solve problems.

**EK 5.5.1J** Basic operations on collections include adding elements, removing elements, iterating over all elements, and determining whether an element is in a collection.

ACM/IEEE-CS Computer Science Curricula 2013: Software Development Fundamentals (SDF)

**SDF/Algorithms and Design Topic** Problem-solving strategies

**SDF/Algorithms and Design Topic** Fundamental design concepts and principles

**SDF/Algorithms and Design LO 3** Create algorithms for solving simple problems

**SDF/Algorithms and Design LO 4** Use a programming language to implement, test, and debug algorithms for solving simple problems.

**SDF/Algorithms and Design LO 8** Apply the techniques of decomposition to break a program into smaller pieces.

**SDF/Fundamental Programming Concepts Topic** Basic syntax and semantics of a higher-level language

**SDF/Fundamental Programming Concepts Topic** Variables and primitive data types

**SDF/Fundamental Programming Concepts Topic** Expressions and assignments

**SDF/Fundamental Programming Concepts Topic** Simple I/O including file I/O

**SDF/Fundamental Programming Concepts Topic** Conditional and iterative structures

**SDF/Fundamental Programming Concepts LO 3** Write programs that use primitive data types.

**SDF/Fundamental Programming Concepts LO 4** Modify and expand short programs that use standard conditional and iterative control structures and functions.

**SDF/Fundamental Programming Concepts LO 5** Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures.

**SDF/Fundamental Programming Concepts LO 6** Write a program that uses file I/O to provide persistence across multiple executions.

**SDF/Fundamental Programming Concepts LO 7** Choose appropriate conditional and iteration constructs for a given programming task.

**SDF/Development Methods Topic** Program correctness, defensive programming

**SDF/Development Methods Topic** Simple refactoring

**SDF/Development Methods LO 3** Identify common coding errors that lead to insecure programs (failure to check inputs) and apply strategies for avoiding such.

# Module Details

**Recommended Time:** 8–9 contact hours

**Prerequisite Knowledge:** Basic knowledge of programming (variables, loops, conditionals); knowledge of a Linux or other UNIX-like system including pipes and input and output redirection. In particular, students should be able to read a manual page with guidance. Instructors should conduct a pre-assessment to verify that students have the above background knowledge. If students do not know these concepts and have no experience with them, much of the module will be more difficult than intended.

**Pilot-tested in the Following Courses:** ITSE 1350, System Analysis and Design; CSC 240, Introduction to Different Programming Languages; NTWK 2013, Introduction to Networking; CIS 215, Operating Systems; CSC 200, Introduction to Computer Science.

**Lab Environment:** Students should have access to the following files:

* “03.SeS\_Unit0\_AllAboutLinux\_Lab.docx” for Unit 0
* “06.SeS\_Unit1\_TheBasics\_Lab.docx” and the contents of the directory “07.SeS\_Unit1\_TheBasics\_DataFiles” for Unit 1
* “12.SeS\_Unit2\_AdvancedControl\_Lab.docx” and the "sample" subdirectory from the directory “13.SeS\_Unit2\_AdvancedControl\_DataFiles” for Unit 2
* “18.SeS\_Unit3\_AdvancedScripting\_Lab.docx” and the file “connect.csv” from the directory “19.SeS\_Unit3\_AdvancedScripting\_DataFiles” for Unit 3

They will also need access to a Linux system, preferably Linux Fedora 22. They will not need Internet or network connectivity once the directories and Laboratory Exercise files are downloaded.

For additional guidance, here are the specific programs that each unit uses. The system used must have them. All are standard, with the possible exception of *shasum*, which produces a cryptographic checksum. If that program is not available, you can replace the occurrence of

shasum $1

with

sum $1 | awk '{ print $1, $3 }'

• Unit 1 uses the commands *bash* (1) or *sh* (1), *grep* (1), *sed* (1), and *awk* (1).

• Unit 2 uses the commands *bash* (1) or *sh* (1), *ls* (1), *shasum* (1), *diff* (1), *rm* (1), *sort* (1), and *uniq* (1).

• Unit 3 uses the commands *bash* (1) or *sh* (1), *date* (1), *expr* (1), *sed* (1), and *tr* (1).

**Homework:** Labs are included in each unit. Lab exercises can be started as a class, during the presentations, and completed as homework assignments. Many of the lab exercises involve writing and later revising BASH scripts. It is recommended that students save each script as a separate file, using the suggested filename provided.

Instructors can check students’ scripts either by running them to test their outputs or by comparing them to the correct answer scripts provided in the directories that end with “ … \_LabSolutionFiles” and “… \_LabSolutionsPDFs.” In the directories ending with “ … \_LabSolutionFiles” are versions of the scripts that have been created as plain text files and usually have the filename extension “.sh”; PDF versions of the same files, which may be easier for instructors to manage, are in the directory “… \_LabSolutionsPDFs.”

**Instructional Files and Online Resources Needed:** PowerPoint slides and data files are included for each unit.

**Assessment:** See the assessment guide for this unit, 23.SeS\_AssessmentGuide. Files containing scripts that are the correct answers to assessment questions are provided in the directory 24.SeS\_AssessmentAnswerFiles. Each script is available in either plain text (“.sh”) format or PDF format, so instructors can use whichever file format they find easier to manage.

### Unit 0. All ABout Linux

**Presentation:** 02.SeS\_Unit0\_AllAboutLinux\_Presentation.pptx

**Lab:** 03.SeS\_Unit0\_AllAboutLinux\_Lab.docx

Solutions to lab questions are located in 04.SeS\_Unit0\_AllAboutLinux\_LabSolutions.docx.

This is unit not intended to be an exhaustive lesson on Linux. It is meant only to provide enough working knowledge of Linux for studients to complete the module.

**Learning Objectives**

Upon completion of Unit 0:

0.1 Students will be able to describe the organization of the Linux file system.

0.2 Students will be able to identify a file using both absolute path names and relative path names (and explain the difference).

0.3 Students will be able to move around the file system using Linux commands.

0.4 Students will be able to explain what the search path is and how it works.

0.5 Students will be able to execute a Linux command.

0.6 Students will be able to demonstrate how to redirect input from and output to a file.

0.7 Students will be able to state what a pipe does and how it relates to input and output redirection.

### Unit 1. The Basics

**Presentation:** 05.SeS\_Unit1\_TheBasics\_Presentation.pptx

To demonstrate the examples included in the presentation, you will need the following data files from the directory 07.SeS\_Unit1\_TheBasics\_DataFiles:

* abcscript — Use for the example shown on Slides 6–7.
* dict.txt — This file contains a list of words, one per line. It’s first used starting on Slide 13.
* add — This script is introduced on Slide 25.
* mycat.sh — This script is discussed on Slides 28–31. It is provided so students can experiment with the different quotation marks.
* x, y, x y — These three files are used with Slides 28-31 to demonstrate the proper use of quotation marks.

If you want the students to try the exercises themselves, copy the files to the students’ systems or make available for them to download.

**Lab:** 06.SeS\_Unit1\_TheBasics\_Lab.docx

Students will need the following data files from the directory 07.SeS\_Unit1\_TheBasics\_DataFiles. These files can be copied to the students’ systems or made available for them to download.

* dict.txt
* mycat.sh
* x
* y
* x y

Solutions to lab questions are located in 08.SeS\_Unit1\_TheBasics\_LabSolutions.docx

**Learning Objectives**

Upon completion of Unit 1:

1.1 Students will be able to analyze a problem and develop a script to solve it.

1.2 Students will be able to create and execute a Bourne shell script.

1.3 Students will be able to use command-line arguments in that script.

1.4 Students will be able to use conditional (*if* … *elif* ... *fi*) statements to test for various conditions and act accordingly.

1.5 Students will be able to perform basic error checking in the script.

### Unit 2. Advanced Control

**Presentation:** 11.SeS\_Unit2\_AdvancedControl\_Presentation.pptx

Files in the directory 13.SeS\_Unit2\_AdvancedControl\_DataFiles are used in the examples.

* for1.sh — This script is an example of a basic *for* loop (see Slide 15).
* myls.sh — This script lists each named file, giving attribute information as well as the filename(s). See Slide 10.
* Subdirectory “sample” — These files are used in the demonstration of *for* loops on Slides 16–18. See notes below under “Lab.”
* for2.sh — This script is an example of a *continue* statement (see Slide 20).
* var1.sh, var2.sh, var3.sh — These show some things about variables. They are referenced on Slides 23–25.
* bool1.sh, bool2.sh — These demonstrate various Boolean operators. See Slides 30–31.
* x1, x2 — These two files containing very similar content are used to demonstrate the output from *diff*. They are referenced on Slide 34.

**Lab:** 12.SeS\_Unit2\_AdvancedControl\_Lab.docx

Students will need a copy of the directory “sample”. It can be copied to the students’ systems or put out for them to download. Please do not change anything in that directory – one of the tests the students run depends on those files being unchanged.

* abc xyz — This file is empty. Note the blank space; that is part of the filename.
* abcde — This file is also empty.
* demofor.sh, demofor2.sh, demofor3.sh — These scripts demonstrate various things about *for* loops.

Solutions to lab questions are located in 14.SeS\_Unit2\_AdvancedControl\_LabSolutions.docx.

**Learning Objectives**

Upon completion of Unit 2:

2.1 Students will be able to merge two or more scripts into one that performs the same functions.

2.2 Students will be able to use *for* loops in the script.

2.3 Students will be able to use variables in the script.

2.4 Students will be able to use Boolean operators to test conditions in the script.

2.5 Students will be able to process options given to the script.

2.6 Students will be able to perform basic error checking in the script.

**Suggestion**

If you want to split this unit into two parts, we recommend using Slides 1 through 21 or 22 as the first part; use Slide 3 followed by Slides 22 through 26 as the second part. There is a natural breaking point after Slide 21, because the class has finished the loops and done an exercise. Slide 22 is a high-level overview of the next few slides and might be useful as a “teaser”; if your students respond well to that sort of slide or discussion ender, go ahead and include Slide 22 at the end of the first part. You should begin the second part with Slide 22, though, to remind students of where the class left off. Also, if you split this unit, we strongly recommend beginning the second part with Slide 3 to remind students of the problem being solved.

### Unit 3. Advanced Scripting

**Presentation:** 17.Ses\_Unit3\_AdvancedScripting\_Presentation.pptx

Files in the directory 19.SeS\_Unit3\_AdvancedScripting\_DataFiles are used in examples during the presentation.

• connect.csv — This is a comma-separated value (CSV) version of a spreadsheet.

• gleep — This is used to demonstrate reading fields from a file. It is referenced on Slides 9 and 10.

* fn1, fn2, round, midmin — These files are only used with the extra material for advanced students, on Slides 35, 36, 37, and 38.

**Lab:** 18.SeS\_Unit3\_AdvancedScripting\_Lab.docx

Students will need the following data file from the directory 19.SeS\_Unit3\_AdvancedScripting\_DataFiles. This file can be copied to the students’ systems or made available for them to download.

* connect.csv — This is a comma-separated value (CSV) version of a spreadsheet.

Solutions to lab questions are located in 20.SeS\_Unit3\_AdvancedScripting\_LabSolutions.docx.

**Learning Objectives**

Upon completion of Unit 3:

3.1 Students will be able to analyze data in a file using a script.

3.2 Students will be able to use while loops in the script.

3.3 Students will be able to do simple arithmetic in the script.

3.4 Students will be able to edit values of variables and data.

3.5 Students will be able to perform pattern matching.

**Note:** The material in Unit 3 includes the use of regular expressions for pattern-matching in programs such as *expr* (1) and *sed* (1). If students have never seen this, the instructor should be prepared to expand on what is in the slides. This may increase the amount of time needed to cover this unit.

**Suggestion**

If you want to split this module into two parts, we recommend using Slides 1 through 22 as the first part and Slides 23 through 46 as the second part. There is a natural breaking point after Slide 22, because the class has finished counting and the *while* loop and done two exercises on the subject. The second part begins with editing variable values, which is different enough that a break here makes sense.