01- Software Development and Design

Ahmed Sultan
Senior Technical Instructor
ahmedsultan.me/about

OUTLINES

1.1- Software Development Lifecycle & SDLC Models and Design Patterns

1.2- Linux Bash

1.3- Software Version Control & Git and GitHub

Labs

1.2 - Linux Bash Commands

1.3 - Git Commands

1.1- Software Development Lifecycle & SDLC Models and Design Patterns

1.2- Linux Bash

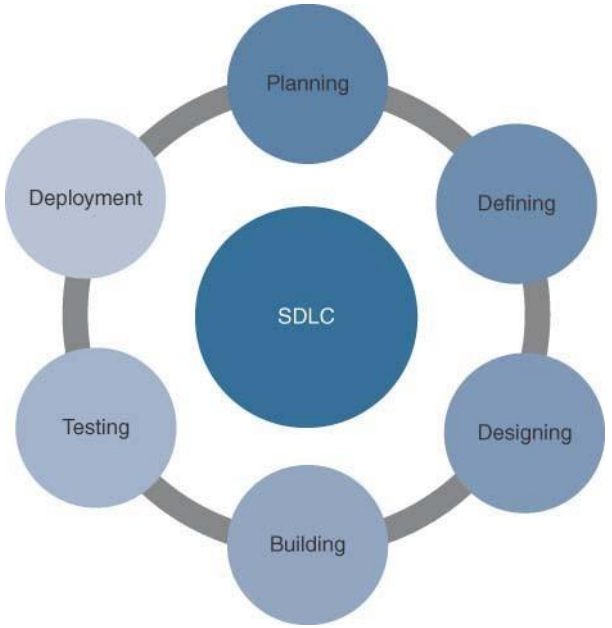
1.3- Software Version Control & Git and GitHub

SOFTWARE DEVELOPMENT LIFECYCLE

- Anyone can program.
- Once you learn a programming language's syntax, it's just a matter of slapping it all together to make your application do what you want it to do.
- The reality is, software needs to be built using a **structure** to give it sustainability, manageability, and coherency.

SOFTWARE DEVELOPMENT LIFECYCLE (cont.)

- **Software Development Lifecycle (SDLC)** provides sanity by providing guidance on building sustainable software.
- **SDLC** lays out a plan for building, fixing, replacing, and making alterations to software.



SOFTWARE DEVELOPMENT LIFECYCLE (cont.)

- These are the stages of the SDLC:
  - ✓Stage 1—Planning: Identify the current use case or problem the software is intended to solve, Get input from stakeholders, end users to determine what success looks like, this stage is also known as **"Requirements Analysis"**
  - ✓Stage 2—Defining: This stage involves analyzing the functional specifications of the software—basically defining what the software is supposed to do.
  - ✓Stage 3—Designing: This is a critical stage as stakeholders need to be in agreement in order to build the software appropriately; if they aren't, users won't be happy, and the project will not be successful.

SOFTWARE DEVELOPMENT LIFECYCLE (cont.)

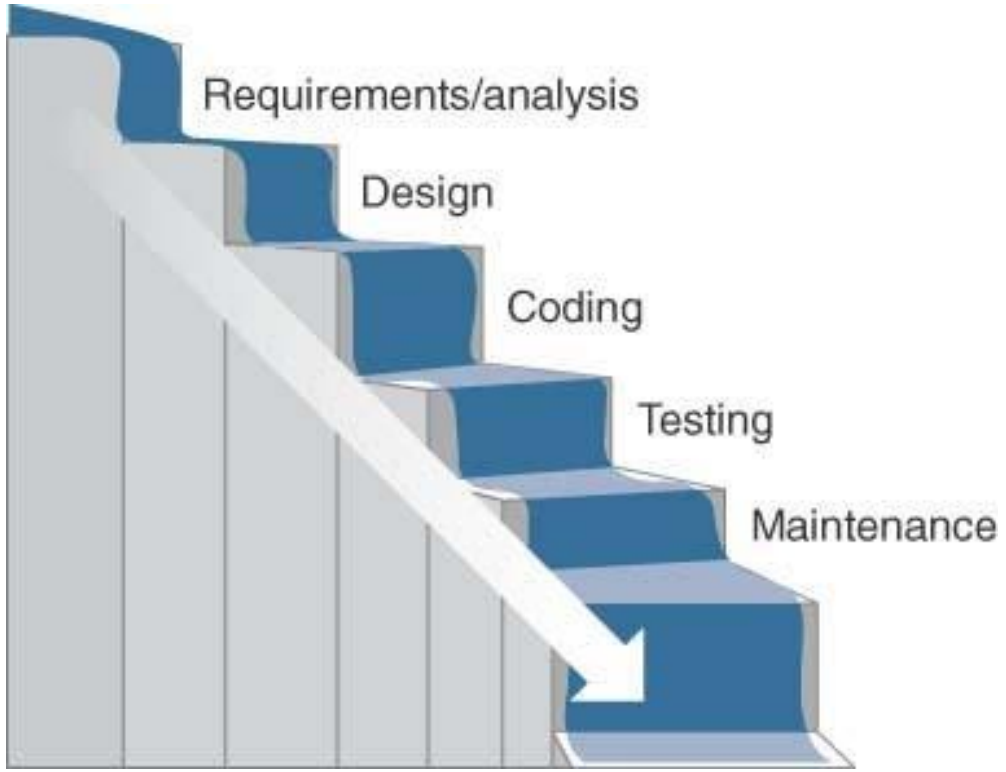
- These are the stages of the SDLC (cont.)
  - ✓Stage 4—Building: Once the software design specification is complete, the programmers get to work on making it a reality.
  - ✓Stage 5—Testing: In this stage the programmers check for bugs and defects, the software continually examined and tested until it successfully meets the original software specifications.
  - ✓Stage 6—Deployment: During this stage, the software is put into production for the end users to put it through its paces.

SOFTWARE DEVELOPMENT LIFECYCLE Models

- There are quite a few SDLC models that further refine the generic process just described.
- They all use the same core concepts but vary in terms of implementation and utility for different projects and teams.
- The following are some of the most popular SDLC models:
  - ✓Waterfall
  - ✓Lean
  - ✓Agile

WATERFALL MODEL

- Waterfall is a serial approach to software development that is divided into phases



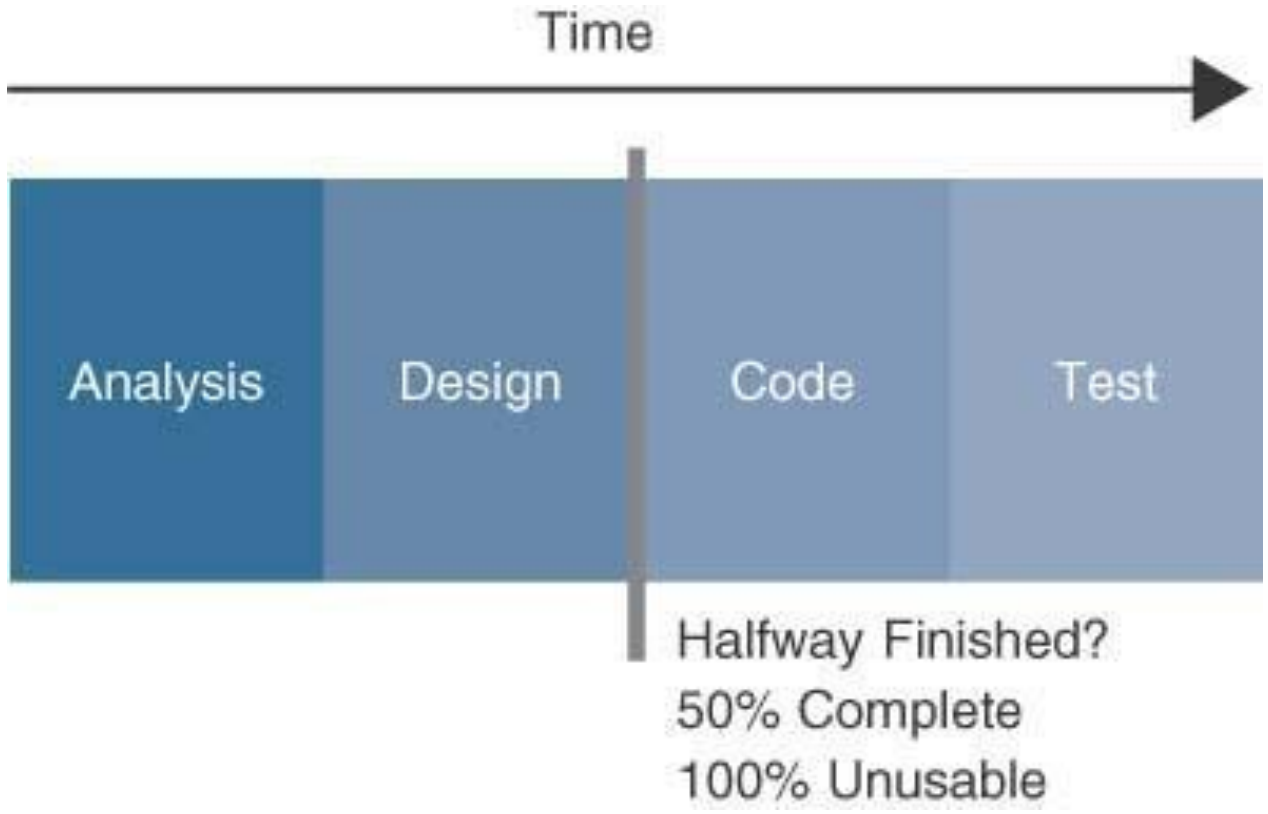
WATERFALL MODEL (cont.)

- Waterfall is a serial approach to software development that is divided into phases:
  1. Requirements/Analysis
  2. Design
  3. Coding
  4. Testing
  5. Maintenance

WATERFALL MODEL (cont.)

- While this approach has worked successfully over the years, a number of shortcomings have become weaknesses in this approach:
  - ✓Making changes to the first floor of a building after you have begun the fifth floor is extremely difficult and may even be impossible unless you knock down the building and start from scratch.
  - ✓Value is not achieved until the end of the whole process, even if you are halfway done with a project, you still have no usable code or value to show to the business
  - ✓When software developers run out of time, testing often suffers or is sacrificed in the name of getting the project out the door.

WATERFALL MODEL (cont.)



LEAN MODEL

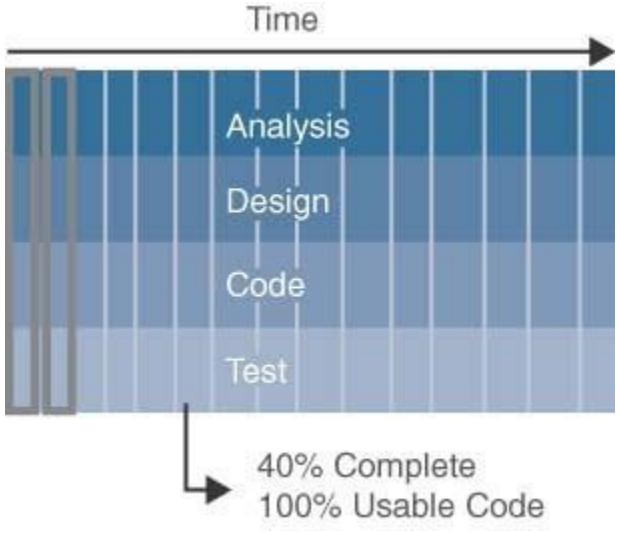
- After **World War II**, Japan was in desperate need of rebuilding.
- Most of Japan's production capabilities had been destroyed, including those in the auto industry.
- When Japan tackled this rebuilding, it didn't concentrate on only the buildings and infrastructure; it looked at ways to do things differently:
  - ✓Elimination of waste: If something doesn't add value to the final product, get rid of it.
  - ✓Just in time: Don't build something until the customer is ready to buy it.
  - ✓Continuous Improvement: Always improve your processes with lesson learned.
- **Lean** led to Agile software development, which has served as a lightning rod of change for IT operations.

AGILE MODEL

- **Agile** is an application of Lean principles to software development.
- Developing software through Agile results in very different output than the slow serial manner used with Waterfall.
- With Waterfall, a project is not “finished” and deployed until the very end.
- With Agile, the time frame is changed, Agile uses smaller time increments **(often 2 weeks)** or “**sprints**” that encompass the full process of analysis, design, code, and test but on a mush smaller aspect of and application.
- The goal is to finish a feature or capability for each sprint, resulting in a potentially shippable incremental piece of software.
- With Agile, if you are **40%** finished with a project, you have **100%** usable code.

AGILE MODEL (cont.)

- By leveraging **Agile**, you can keep adding value immediately and nimbly adapt to change.
- If a new capability is needed in the software, or if a feature that was planned is determined to no longer be necessary, the project can pivot quickly and make those adjustments.



COMMON DESIGN PATTERNS

- When creating software, you will often run into the same problem over and over and over again.
- You don’t have to reinvent the wheel each time you need a rolling thing to make something move.
- Many common design paradigms have already been created, and you can reuse them in your software project.
- These design patterns make you faster and provide true solutions that have been tested and refined.

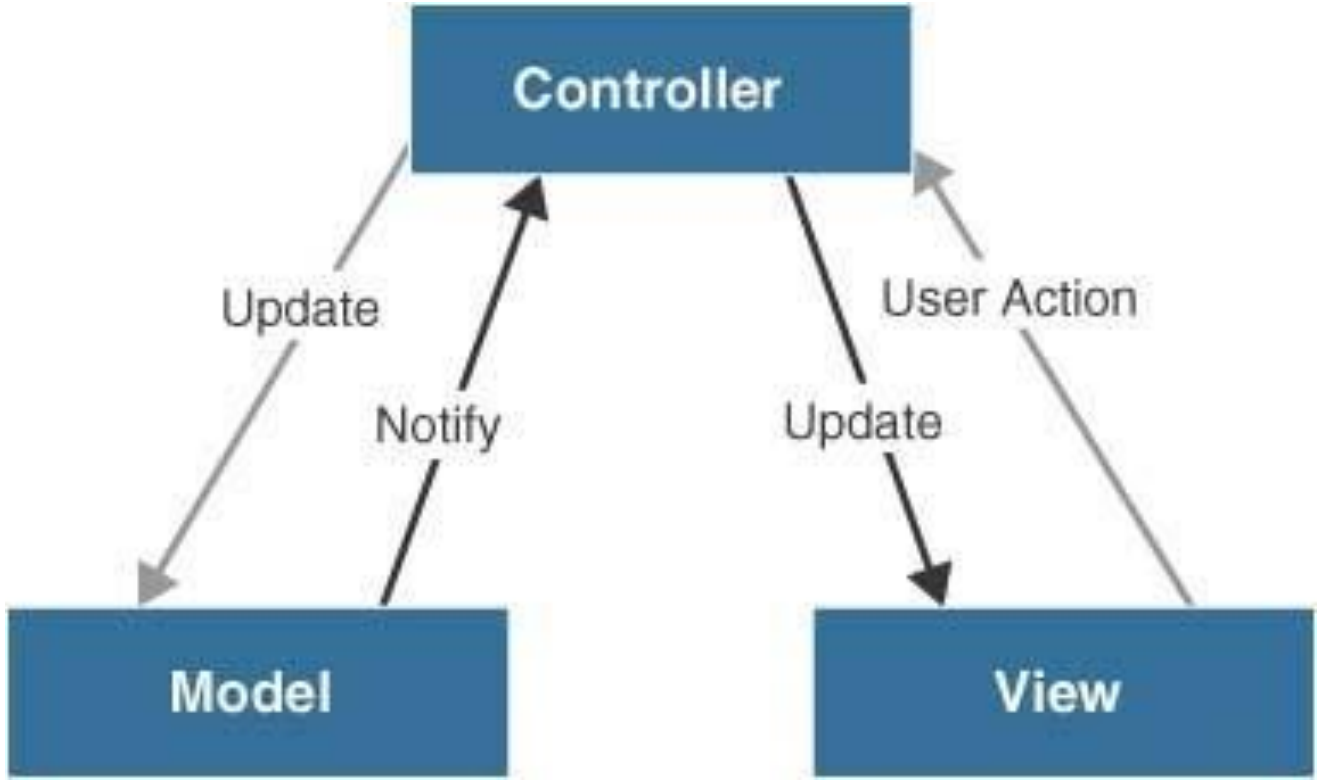
MODEL-VIEW-CONTROLLER (MVC) PATTERN

- The **Model-View-Controller (MVC)** pattern was one of the first design patterns to leverage the **separation of concerns (SoC)** principle.
- The **SoC** principle is used to decouple an application's interdependencies and functions from its other parts.
- The goal is to make the various layers of the application—such as data access, business logic, and presentation (to the end user)— modular.
- This modularity makes the application easier to construct and maintain while also allowing the flexibility to make changes or additions to business logic.

MODEL-VIEW-CONTROLLER (MVC) PATTERN (cont.)

- The classical **MVC** pattern has three main parts:
  - ✓Model: The model is responsible for retrieving and manipulating data, It is often tied to some type of database but could be data from a simple file.
  - ✓View: The view is what the end users see on the devices they are using to interact with the program.
  - ✓Controller: The Controller is the intermediary between what the user sees and the backend logic that manipulates the data, The role of the controller is to receive requests from the user via the view and pass those requests on the model and its underlying data store.

MODEL-VIEW-CONTROLLER (MVC) PATTERN (cont.)



1.1- Software Development Lifecycle & SDLC Models and Design Patterns

1.2- Linux Bash

1.3- Software Version Control & Git and GitHub

LINUX BASH

- Knowing how to use Linux **BASH** is a necessary skill for working with open-source technologies as well as many of the tools you need to be proficient with to be successful in the development world.
- Linux has taken over the development world, and even Microsoft has jumped into the game by providing the **Windows Subsystem for Linux (WSL)** for Windows 10 pro.
- For the DEVASC exam, you need to know how to use **BASH** and be familiar with some of the key commands.

LINUX BASH (cont.)

- **BASH** is a shell, and a shell is simply a layer between a user and the internal workings of an operating system.
- A user can use the shell to input commands that the operating system will interpret and perform.
- Before graphical user interfaces (GUI) became common, the shell reigned supreme.
- While there are many shells you can use, **BASH**, which stands for **Bourne Again Shell**, is one of the most popular.
- It has been around since 1989 and is the default shell on most Linux operating systems.

LINUX BASH (cont.)

- Until recently, it was also the default for the **Mac operating system**, but Apple has replaced BASH with **Z shell**.
- The commands and syntax you will learn with **BASH** are transferable to **Z shell**, as it was built to maintain compatibility with BASH.
- BASH is not only a shell for command processing using standard Linux operating system commands.
- It can also read and interpret scripts for automation.

LAB

Install BASH and get familiar with BASH commands

1.1- Software Development Lifecycle & SDLC Models and Design Patterns

1.2- Linux Bash

1.3- Software Version Control & Git and GitHub

SOFTWARE VERSION CONTROL

- The term **version control** is used to describe the process of saving various copies of a file or set of files in order to track changes made to those files.
- **Software Version Control (SVC)** typically involves a database that stores current and historical versions of source code to allow multiple people or teams you want to go back a revision (or to any previous version).
- If you have a team of developers working on a project, all writing new code or changing previously written code, losing those files could set you back weeks or months.
- Version control can protect your code by allowing changes to be checked in (through a process known as a **code commit**) to a hierarchical tree structure of folders with files in them.

SOFTWARE VERSION CONTROL (cont.)

- Each check-in is tagged with who made the change and what the person changed within the code.
- Instead of using inefficient techniques such as file locking, a version control system handles concurrent check-ins, allowing two programmers to commit code at the same time.
- Another aspect of a version control system is the ability to **branch** and **merge** code built independently.
- This is very useful if you are writing code on a part of an application the could conflict with another part written by another team.
- By creating a branch, you effectively create a separate work stream that has its own history and does not impact the main "**trunk**" of the code base.

SOFTWARE VERSION CONTROL (cont.)

- Can you write software without a version control system? Sure, but why would you? A lot of version control software options are available, many of them free, and it is good practice to always use a version control system to store your code.
- **GIT** is one of the most commonly used version control systems today.

GIT

- A staggering number of companies use Git, which is free and open source.
- **Git** was created to be fast and scalable, with a distributed workflow that could support the huge number of contributors to the Linux kernel.
- Note
  - **GitHub is not Git**, GitHub is a cloud-based social networking platform for programmers that allows anyone to share and contribute to software projects.
  - While GitHub uses Git as its version control system underneath the graphical front end, it is not directly tied to the Git open-source project (much as a Linux distribution, such as Ubuntu or Fedora, uses the Linux kernel but is independently developed from it).

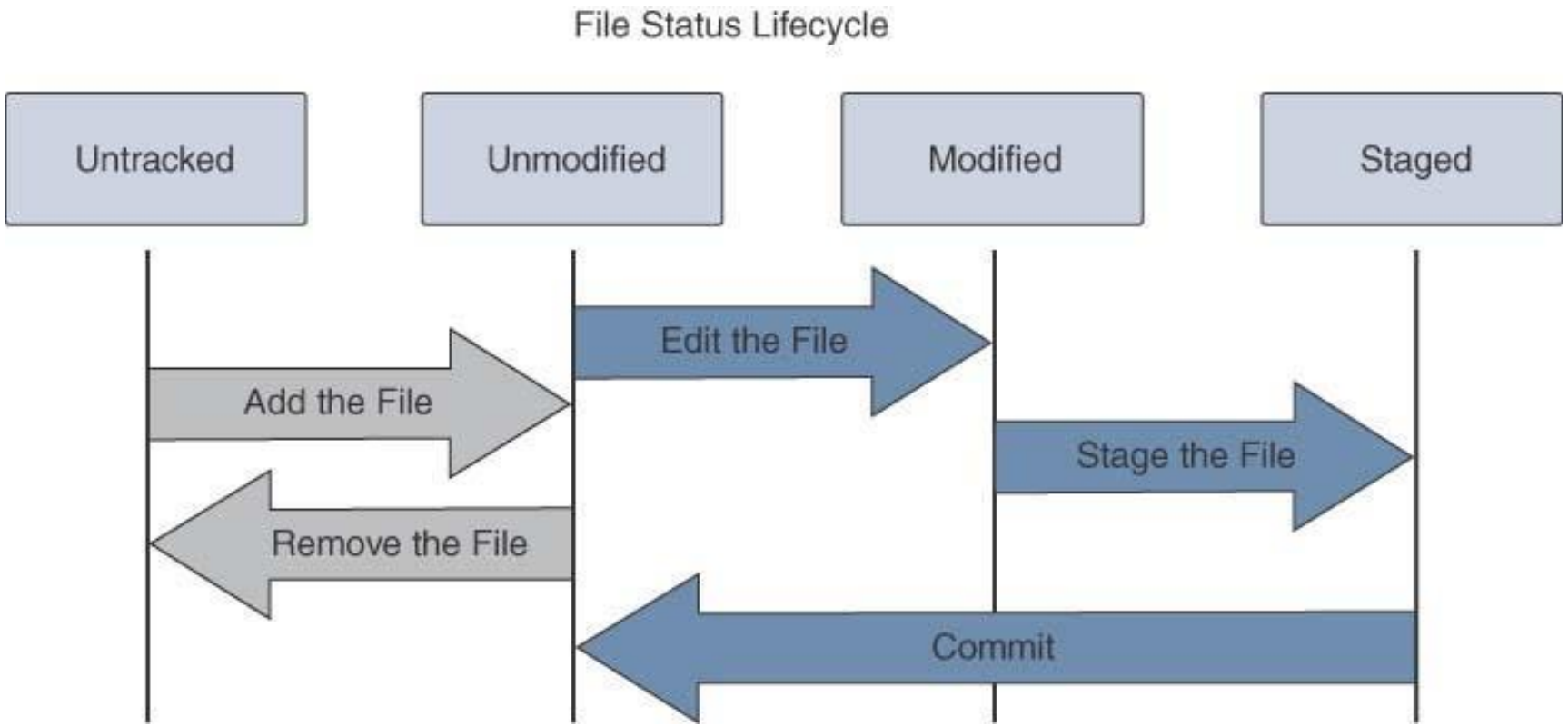
GIT (cont.)

- **Git** is a distributed version control system built with scalability in mind.
- It uses a multi-tree structure, and if you look closely at the design, you see that it looks a lot like a file system.
- **Git** keeps track of three main structures, or trees:
  - ✓Local workspace: This is where you store source code files, binaries, images and documentations.
  - ✓Staging area: This is intermediary storage area for items to be synchronized (changes and new items)
  - ✓Head, or local repository: This is where you store all committed items.

GIT (cont.)

- Each file that you add to your working directory has a status attributed to it:
  - ✓Untracked: when you first create a file in a directory that Git is managing, it is given an untracked status, if you want Git to start tracking a file, you have to explicitly tell it to do so with **git add** command.
  - ✓Unmodified: A tracked file in Git is included as part of the repository, and changes are watched, This status means Git is watching for any file changes that are made but it doesn't see any yet.
  - ✓Modified: Whenever you add some code or make a change to the file, Git changes the status of the file to modified, you have to tell Git that you are ready to add a modified file to the index or staging are are by issuing the **git add** command again.
  - ✓Staged: Once a changed file is added to the index, Git needs to able to bundle up your changes and update the local repository, This process can be accomplished through **git commit** command.

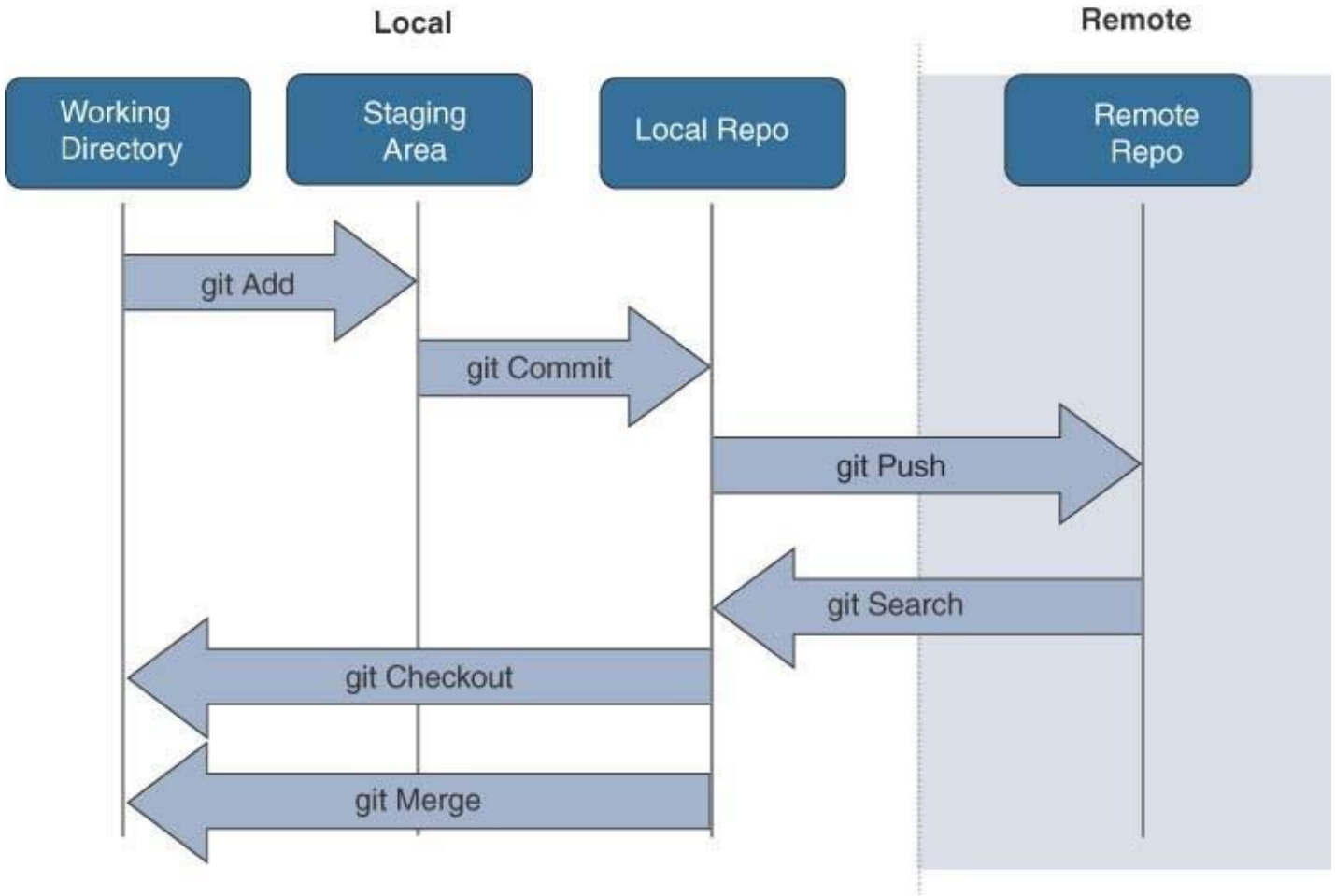
GIT (cont.)



GIT (cont.)

- If at any point you want to see the status of a file from your repository, you can use the extremely useful command **git status** to learn the status of each file in your local directory.
- You can pull files and populate your working directory for a project that already exists by making a clone.
- Once you have done this, your working directory will be an exact match of what is stored in the repository.
- **The next step** is to perform a commit and package up the changes for submission (or pushing) to the remote repository (usually a server somewhere local or on the Internet).

GIT (cont.)


Local
Remote
Working Directory
Staging Area
Local Repo
Remote Repo
git Add
git Commit
git Push
git Search
git Checkout
git Merge

GIT (cont.)

- **Git** may not come natively with your operating system.
- If you are running a Linux variation, you probably already have it.
- For Mac and Windows you need to install it.
- You can go to the main distribution website (https://git- scm.com) and download builds for your operating system directly.
- You can install the command-line version of Git and start using and practicing the commands.
- There are also GUI-based Git clients.

LAB

Install GIT and get familiar with GIT commands