

CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY USING DevOps

Exercise 1:

Get an understanding of the stages in software development lifecycle, the process models, values and principles of agility and the need for agile software development. This will enable you to work in projects following an agile approach to software development.

Defining SDLC

A systematic approach that generates a structure for the developer to design, create and deliver high-quality software based on customer requirements and needs. The primary goal of the SDLC process is to produce cost-efficient and high-quality products. The process comprises a detailed plan that describes how to develop, maintain, and replace the software.

Phases Of SDLC (Software Development Life Cycle)

Stage 1: Project Planning

The first stage of SDLC is all about “What do we want?” Project planning is a vital role in the software delivery lifecycle since this is the part where the team estimates the cost and defines the requirements of the new software.

Stage 2: Gathering Requirements & Analysis

The second step of SDLC is gathering maximum information from the client requirements for the product. Discuss each detail and specification of the product with the customer. The development team will then analyze the requirements keeping the design and code of the software in mind. The main goal of this stage is that everyone understands even the minute detail of the requirement. Hardware, operating systems, programming, and security are to name the few requirements.

Stage 3: Design

In the design phase (3rd step of SDLC), the program developer scrutinizes whether the prepared software suffices all the requirements of the end-user. Additionally, if the project is feasible for the customer technologically, practically, and financially. Once the developer decides on the best design approach, he then selects the program languages like Oracle, Java, Python, C, C++, etc., that will suit the software.

Once the design specification is prepared, all the stakeholders will review this plan and provide their feedback and suggestions. It is absolutely mandatory to collect and incorporate stakeholder’s input in the document, as a small mistake can lead to cost over run.

Stage 4: Coding or Implementation

Time to code! It means translating the design to a computer-legible language. In this fourth stage of SDLC, the tasks are divided into modules or units and assigned to various developers. The developers will then start building the entire system by writing code using the programming languages they chose. This stage is considered to be one of the longest in SDLC. The developers need certain predefined coding guidelines, and programming tools like interpreters, compilers, debugger to implement the code.

Stage 5: Testing

Once the developers build the software, then it is deployed in the testing environment. Then the testing team tests the functionality of the entire system. In this fifth phase of SDLC, the testing is done to ensure that the entire application works according to the customer requirements.

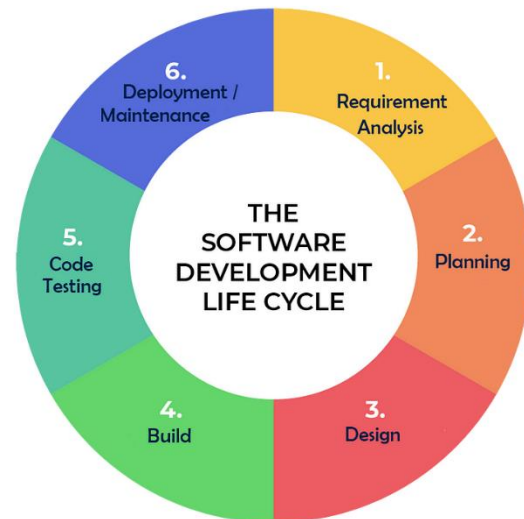
After testing, the QA and testing team might find some bugs or defects and communicate the same with the developers. The development team then fixes the bugs and send it to QA for a re-test. This process goes on until the software is stable, bug-free and working according to the business requirements of that system.

Stage 6: Deployment

The sixth phase of SDLC: Once the testing is done, and the product is ready for deployment, it is released for customers to use. The size of the project determines the complexity of the deployment. The users are then provided with the training or documentation that will help them to operate the software. Again, a small round of testing is performed on production to ensure environmental issues or any impact of the new release.

Stage 7: Maintenance

The actual problem starts when the customer actually starts using the developed system and those needs to be solved from time to time. Maintenance is the seventh phase of SDLC where the developed product is taken care of. According to the changing user end environment or technology, the software is updated timely.



Software process model

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the **order of activities** of the process and the **sequence** in which they are performed.

The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.

There are many kinds of process models for meeting different requirements. We refer to these as **SDLC models** (Software Development Life Cycle models). The most popular and important SDLC models are as follows:

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Prototype model
- Spiral model

- **Project requirements**
- **Project size**
- **Project complexity**
- **Cost of delay**
- **Customer involvement**
- **Familiarity with technology**
- **Project resources**

Agile model

The agile process model encourages **continuous iterations of development** and testing. Each incremental part is developed over an iteration, and each iteration is designed to be small and manageable so it can be completed within a few weeks.

Each iteration focuses on implementing a small set of features completely. It involves customers in the development process and minimizes documentation using informal communication.

Agile development considers the following:

- Requirements are assumed to change
- The system evolves over a series of short iterations
- Customers are involved during each iteration
- Documentation is done only when needed

Though agile provides a very realistic approach to software development, it isn't great for complex projects. It can also present challenges during transfers as there is very little documentation. Agile is great for projects with **changing requirements**.

Some commonly used agile methodologies include:

- **Scrum:** One of the most popular agile models, Scrum consists of iterations called sprints. Each sprint is between 2 to 4 weeks long and is preceded by planning. You cannot make changes after the sprint activities have been defined.
- **Extreme Programming (XP):** With Extreme Programming, an iteration can last between 1 to 2 weeks. XP uses pair programming, continuous integration, test-driven development and test automation, small releases, and simple software design.
- **Kanban:** Kanban focuses on visualizations, and if any iterations are used they are kept very short. You use the Kanban Board that has a clear representation of all project activities and their numbers, responsible people, and progress.

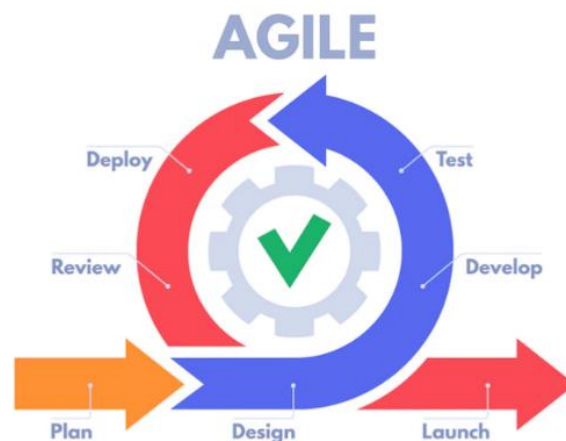
What will enable you to work in projects following an agile approach to software development.

Jira and Agile Projects

Atlassian's Jira has emerged as the tool of choice for managing agile projects in the enterprise. This class teaches you to align agile practices, roles, and ceremonies with the Jira environment. Return to work ready to use Jira to manage technology projects, IT service delivery, and agile software development.

Use Jira for executing everyday agile project work.

This fast-paced, hands-on workshop teaches you how to manage agile projects, products, and development work using Atlassian Jira as your primary collaboration platform. In Jira, you have a tool designed to enable agile practices and track agile projects across a wide range of scenarios. **Features such as issue tracking, search, reporting, and customization allow for seamless integration of agile projects in one project management environment.**



Exercise 2:

Get a working knowledge of using extreme automation through XP programming practices of test first development, refactoring and automating test case writing

What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software. eXtreme Programming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the **Agile software development** methodologies. **It provides values and principles to guide the team behavior.** The team is expected to self-organize.

Extreme Programming provides specific core practices where –

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

This can be achieved with –

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

Extreme Programming involves –

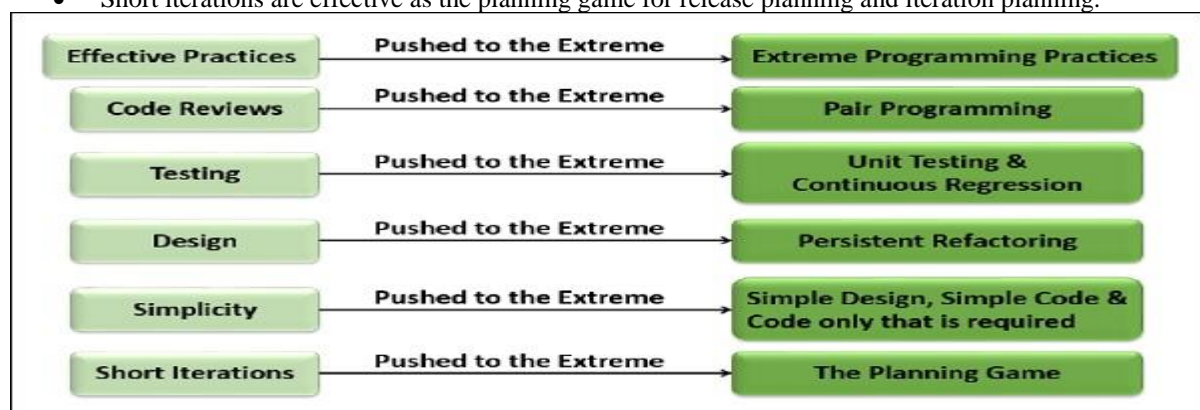
- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.
- Starting with a simple design just enough to code the features at hand and redesigning when required.
- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
- Integrating and testing the whole system several times a day.
- Putting a minimal working system into the production quickly and upgrading it whenever required.
- Keeping the customer involved all the time and obtaining constant feedback.

Iterating facilitates the accommodating changes as the software evolves with the changing requirements.

Why is it called “Extreme?”

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.



Extreme Programming Values

Extreme Programming (XP) is based on the five values –

- Communication
- Simplicity
- Feedback
- Courage
- Respect

In Extreme Programming, the basic principles are derived from the values so that the development practices can be checked against these principles. Each principle embodies the values and is more concrete, i.e. rapid feedback – you either, have it or you do not.

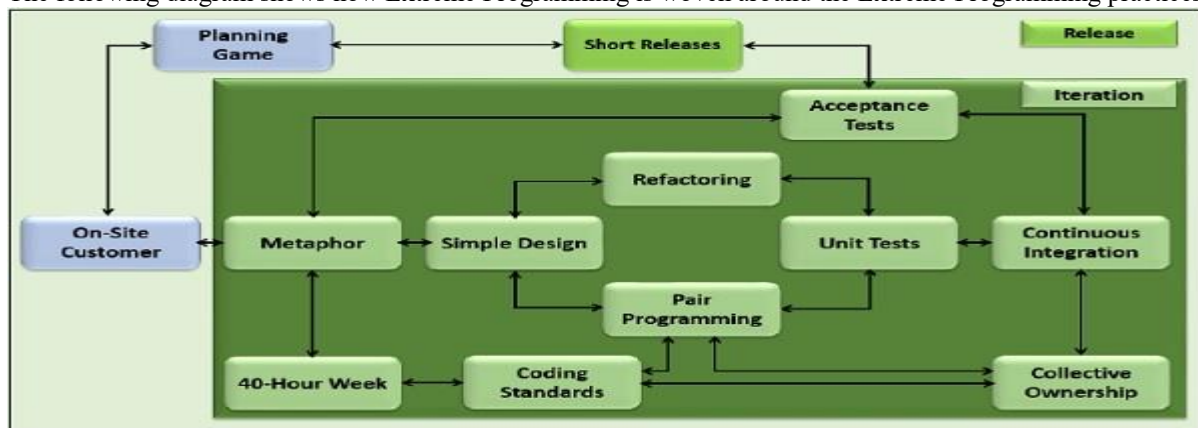
- **Rapid feedback**
- **Assume simplicity**
- **Incremental change**
- **Embracing change**
- **Quality work**

- **Coding**
- **Testing**
- **Listening**
- **Designing**

The Extreme Programming practices can be grouped into four areas –

- **Rapid, Fine Feedback –**
 - **Testing**
 - **On-site customer**
 - **Pair programming**
- **Continuous Process –**
 - **Continuous Integration**
 - **Refactoring**
 - **Short Releases**
- **Shared Understanding –**
 - **The Planning Game**
 - **Simple Design**
 - **Metaphor**
 - **Collective Ownership**
 - **Coding Standards**
- **Developer Welfare –**
 - **Forty-Hour Week**

The following diagram shows how Extreme Programming is woven around the Extreme Programming practices –



There are two roles in each pair –

- The pairing is dynamic. It means that the two Roles A and B may exchange their places, or they may pair up with other team members. More often, anyone on the team will do as a partner. For example, if you have a responsibility for a task in an area that is unfamiliar to you, you might ask someone with recent experience to pair with you.

Continuous Integration

Code is integrated and tested many times a day, one set of changes at a time. A simple way to do this is to have a machine dedicated to integration. A pair with code ready to integrate –

- Sits when the machine is free.
- Loads the current release.
- Loads their changes (checking for and resolving any collisions).
- Runs the tests until they pass (100% correct).

Continuous Integration – Advantages

The advantages of Continuous Integration are –

- Reduces the duration, which is otherwise lengthy.
- Enables the short releases practice as the time required before release is minimal.

40-Hour Week

Extreme Programming emphasizes on the limited number of hours of work per week for every team members, based on their sustainability, to a maximum of 45 hours a week. If someone works for more time than that, it is considered as overtime. Overtime is allowed for at most one week. This practice is to ensure that every team member be fresh, creative, careful and confident.

40-Hour Week – Advantages

The advantages of 40-hour week are –

- Most developers lose effectiveness past 40 hours.
- Value is placed on the developers' well-being.
- Management is forced to find real solutions.

On-Site Customer

Include a real, live user on the team, available full-time to answer the questions, resolve disputes and set small-scale priorities. This user may not have to spend 40 hours on this role only and can focus on other work too.

On-Site Customer – Advantages

The advantages of having an onsite customer are –

- Can give quick and knowledgeable answers to the real development questions.
- Makes sure that what is developed is what is needed.
- Functionality is prioritized correctly.

Coding Standards

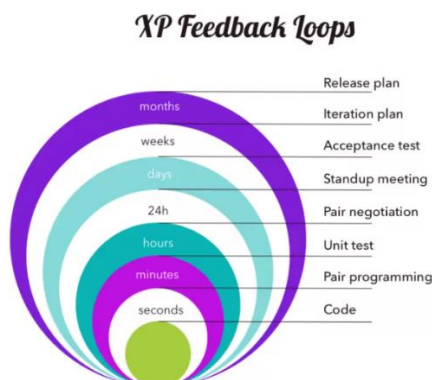
Developers write all code in accordance with the rules emphasizing-

- Communication through the code.
- The least amount of work possible.
- Consistent with the “once and only once” rule (no duplicate code).
- Voluntary adoption by the whole team.

These rules are necessary in Extreme Programming because all the developers –

- Can change from one part of the system to another part of the system.
- Swap partners a couple of times a day.
- Refactor each other's code constantly.

If the rules are not followed, the developers will tend to have different sets of coding practices, the code becomes inconsistent over time and it becomes impossible to say who on the team wrote what code.



When to use XP

Now that we discussed the XP methodology pros and cons and identified its place among other agile frameworks, we can talk about the cases when it's applicable. It's important to make sure a company's size, structure, and expertise, as well as the staff's knowledge base allow for applying XP practices. These are the factors to consider.

1.Highly-adaptive development. 2.Risky projects. 3.Small teams. 4. Automated testing. 5. Readiness to accept new culture and knowledge. 6. Customer participation.

Exercise 3:

It is important to comprehend the need to automate the software development lifecycle stages through DevOps. Gain an understanding of the capabilities required to implement DevOps, continuous integration and continuous delivery practices.

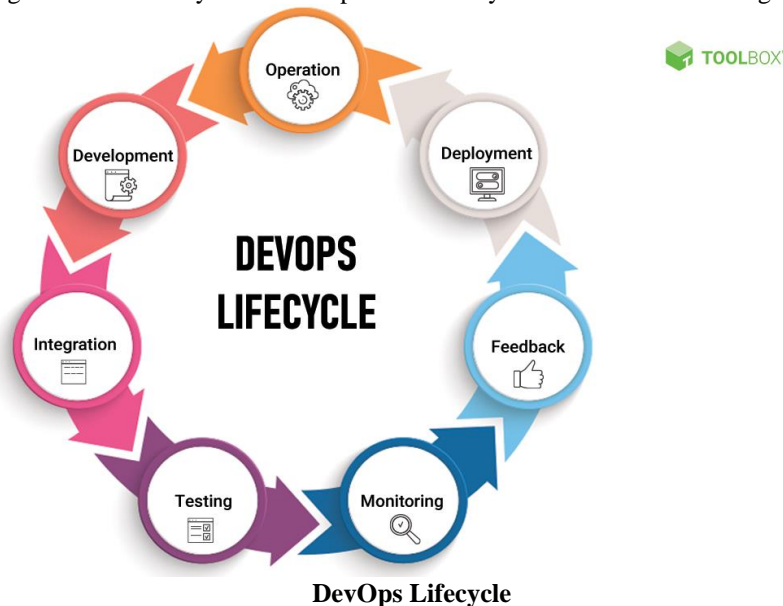
What Is DevOps Lifecycle?

DevOps lifecycle is a combination of different phases of continuous software development, integration, testing, deployment, and monitoring. A competent DevOps lifecycle is necessary to leverage the full benefits of the DevOps methodology.

To deliver faster results, developers must be fully aware of all the different phases of the DevOps lifecycle. If they aren't, the entire development process can become complex and time-consuming. Here is a complete breakdown and analysis of each component of the DevOps lifecycle.

DevOps Lifecycle: Key Components

The DevOps lifecycle optimizes development processes from start to end and engages the organization in continuous development, resulting in faster delivery times. This process mainly consists of the following seven stages.



1. Continuous development

Continuous development, “like agile, began as a software development methodology. Rather than improving software in one large batch, updates are made continuously, piece-by-piece, enabling software code to be delivered to customers as soon as it is completed and tested.

Continuous development involves planning and coding the software. Here, the entire development process gets broken down into smaller development cycles. This process makes it easier for the DevOps team to accelerate the overall software development process.

2. Continuous integration

Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. It’s a primary DevOps best practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run. Automated tools are used to assert the new code’s correctness before integration.

Continuous integration (CI) includes different steps related to the execution of the test process. Along with this, clients also provide information to be incorporated for adding new features to the application. Most changes happen in the source code during this phase. CI becomes the hub for resolving these frequent changes on a daily or monthly basis.

3. Continuous testing

The DevOps lifecycle is the testing phase, wherein the developed code is tested for bugs and errors that may have made their way into the code. This is where quality analysis (QA) plays a major role in checking the usability of the developed software. Successful completion of the QA process is crucial in determining whether the software meets the client’s specifications.

Automation tools, such as JUnit, Selenium, and TestNG, are used for continuous testing, enabling the QA team to analyze multiple code-bases simultaneously. Doing this ensures that there are no flaws in the functionality of the developed software.

4. Continuous deployment

Continuous delivery is an approach where teams release quality products frequently and predictably from source code repository to production in an automated fashion.

Continuous deployment (CD) ensures hassle-free product deployment without affecting the application's performance. It is necessary to ensure that the code is deployed precisely on all available servers during this phase. This process eliminates the need for scheduled releases and accelerates the feedback mechanism, allowing developers to address issues more quickly and with greater accuracy.

5. Continuous monitoring

Monitoring the performance of a software product is essential to determine the overall efficacy of the product output. This phase processes important information about the developed app. Through continuous monitoring, developers can identify general patterns and gray areas in the app where more effort is required.

Continuous monitoring is an operational phase where the objective is to enhance the overall efficiency of the software application. Moreover, it monitors the performance of the app as well. Therefore, it is one of the most crucial phases of the DevOps lifecycle.

Different system errors such as 'server not reachable', 'low memory', etc., are resolved in the continuous monitoring phase. It also maintains the availability and security of the services. Network issues and other problems are automatically fixed during this phase at the time of their detection.

Tools such as Nagios, Splunk, Sensu, ELK Stack, and NewRelic are used by the operations team to monitor user activities for improper behavior.

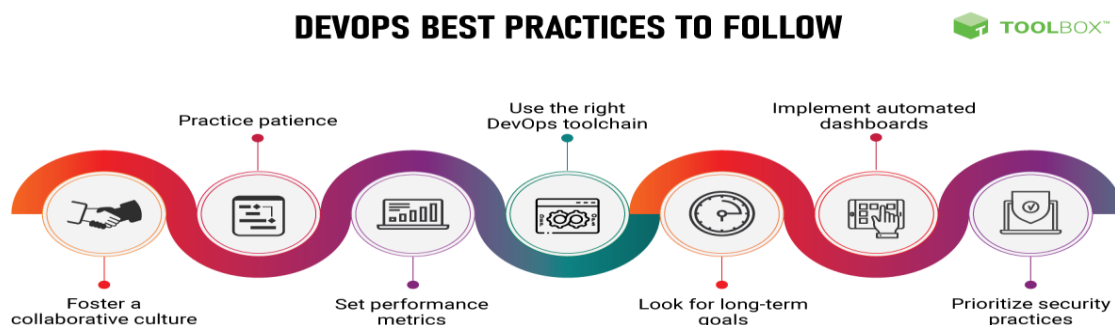
6. Continuous feedback

Continuous feedback is essential to ascertain and analyze the final outcome of the application. It sets the tone for improving the current version and releasing a new version based on stakeholder feedback.

The overall process of app development can only be improved by analyzing the results from the software operations. Feedback is nothing but information gathered from the client's end. Here, information is significant, as it carries all the data about the performance of the software and its related issues. It also contains suggestions given by end users of the software.

7. Continuous operations

The last stage in the DevOps lifecycle is the shortest and easiest to grasp. Continuity is at the heart of all DevOps operations that helps automate release processes, allows developers to detect issues quickly, and build better versions of software products. Continuation is key to eliminate diversions and other extra steps that hinder development. Development cycles in continuous operations are shorter, allowing organizations to advertise constantly and accelerate the overall time to market the product. [DevOps](#) enhances the value of software products by making them better and more efficient, thereby attracting new customers towards it.



DevOps capabilities

The DevOps Research and Assessment (DORA) team has identified and validated a set of capabilities that drive higher software delivery and organizational performance. These articles describe how to implement, improve, and measure these capabilities.

Technical capabilities

Cloud infrastructure

Find out how to manage cloud infrastructure effectively so you can achieve higher levels of agility, availability, and cost visibility.

Code maintainability

Make it easy for developers to find, reuse, and change code, and keep dependencies up-to-date.

Continuous delivery

Make deploying software a reliable, low-risk process that can be performed on demand at any time.

Continuous integration

Learn about common mistakes, ways to measure, and how to improve on your continuous integration efforts.

Continuous testing

Improve software quality by building reliable automated test suites and performing all kinds of testing throughout the software delivery lifecycle.

Database change management

Make sure database changes don't cause problems or slow you down.

Deployment automation

Best practices and approaches for deployment automation and reducing manual intervention in the release process.

Empowering teams to choose tools

Empower teams to make informed decisions on tools and technologies. Learn how these decisions drive more effective software delivery.

Loosely coupled architecture

Learn about moving from a tightly coupled architecture to service-oriented and microservice architectures without re-architecting everything at once.

Monitoring and observability

Learn how to build tooling to help you understand and debug your production systems.

Shifting left on security

Build security into the software development lifecycle without compromising delivery speed.

Test data management

Understand the right strategies for managing test data effectively along with approaches to provide fast, secure data access for testing.

Trunk-based development

Prevent merge-conflict hassles with trunk-based development practices.

Version control

A guide to implementing the right version control practices for reproducibility and traceability.

Process capabilities**Customer feedback**

Drive better organizational outcomes by gathering customer feedback and incorporating it into product and feature design.

Monitoring systems to inform business decisions

Improve monitoring across infrastructure platforms, middleware, and the application tier, so you can provide fast feedback to developers.

Proactive failure notification

Set proactive failure notifications to identify critical issues and act on problems before they arise.

Streamlining change approval

Replace heavyweight change-approval processes with peer review, to get the benefits of a more reliable, compliant release process without sacrificing speed.

Team experimentation

Innovate faster by building empowered teams that can try out new ideas without approval from people outside the team.

Visibility of work in the value stream

Understand and visualize the flow of work from idea to customer outcome in order to drive higher performance.

Visual management capabilities

Learn about the principles of visual management to promote information sharing, get a common understanding of where the team is, and how to improve.

Work in process limits

Prioritize work, limit the amount of things that people are working on, and focus on getting a small number of high-priority tasks done.

Working in small batches

Create shorter lead times and faster feedback loops by working in small batches. Learn common obstacles to this critical capability and how to overcome them.

Cultural capabilities**Generative organizational culture**

Discover how growing a generative, high-trust culture drives better organizational and software delivery performance.

Job satisfaction

Find out about the importance of ensuring your people have the tools and resources to do their job, and of making good use of their skills and abilities.

Learning culture

Grow a learning culture and understand its impact on your organizational performance.

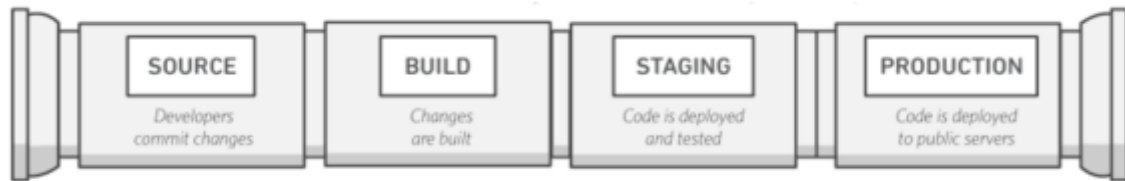
Transformational leadership

Learn how effective leaders influence software delivery performance by driving the adoption of technical and product management capabilities.

Practicing Continuous Integration and Continuous Delivery

Implementing continuous integration and continuous delivery

A pathway to continuous integration/continuous delivery CI/CD can be pictured as a pipeline (refer to the following figure), where new code is submitted on one end, tested over a series of stages (source, build, staging, and production), and then published as production-ready code. If your organization is new to CI/CD it can approach this pipeline in an iterative fashion. This means that you should start small, and iterate at each stage so that you can understand and develop your code in a way that will help your organization grow.



CI/CD pipeline

Each stage of the CI/CD pipeline is structured as a logical unit in the delivery process. In addition, each stage acts as a gate that vets a certain aspect of the code. As the code progresses through the pipeline, the assumption is that the quality of the code is higher in the later stages because more aspects of it continue to be verified. Problems uncovered in an early stage stop the code from progressing through the pipeline. Results from the tests are immediately sent to the team, and all further builds and releases are stopped if software does not pass the stage.

These stages are suggestions. You can adapt the stages based on your business need. Some stages can be repeated for multiple types of testing, security, and performance. Depending on the complexity of your project and the structure of your teams, some stages can be repeated several times at different levels. For example, the end product of one team can become a dependency in the project of the next team. This means that the first team's end product is subsequently staged as an artifact in the next team's project.

The presence of a CI/CD pipeline will have a large impact on maturing the capabilities of your organization. The organization should start with small steps and not try to build a fully mature pipeline, with multiple environments, many testing phases, and automation in all stages at the start. Keep in mind that even organizations that have highly mature CI/CD environments still need to continuously improve their pipelines.

Building a CI/CD-enabled organization is a journey, and there are many destinations along the way. The next section discusses a possible pathway that your organization could take, starting with continuous integration through the levels of continuous delivery.

Continuous integration



The first phase in the CI/CD journey is to develop maturity in continuous integration. You should make sure that all of the developers regularly commit their code to a central repository (such as one hosted in CodeCommit or GitHub) and merge all changes to a release branch for the application. No developer should be holding code in isolation. If a feature branch is needed for a certain period of time, it should be kept up to date by merging from upstream as often as possible. Frequent commits and merges with complete units of work are recommended for the team to develop discipline and are encouraged by the process. A developer who merges code early and often, will likely have fewer integration issues down the road.

You should also encourage developers to create unit tests as early as possible for their applications and to run these tests before pushing the code to the central repository. Errors caught early in the software development process are the cheapest and easiest to fix.

When the code is pushed to a branch in a source code repository, a workflow engine monitoring that branch will send a command to a builder tool to build the code and run the unit tests in a controlled environment. The build process should be sized appropriately to handle all activities, including pushes and tests that might happen during the commit stage, for fast feedback. Other quality checks, such as unit test coverage, style check, and static analysis, can happen at this stage as well. Finally, the builder tool creates one or more binary builds and other artifacts, like images, stylesheets, and documents for the application.

Continuous delivery: creating a staging environment



Continuous delivery—staging

Continuous delivery (CD) is the next phase and entails deploying the application code in a staging environment, which is a replica of the production stack, and running more functional tests. The staging environment could be a static environment premade for testing, or you could provision and configure a dynamic environment with committed infrastructure and configuration code for testing and deploying the application code.

Continuous delivery: creating a production environment



Continuous delivery—production

In the deployment/delivery pipeline sequence, after the staging environment, is the production environment, which is also built using infrastructure as code (IaC).

Continuous deployment



Continuous deployment

The final phase in the CI/CD deployment pipeline is continuous deployment, which may include full automation of the entire software release process including deployment to the production environment. In a fully mature CI/CD environment, the path to the production environment is fully automated, which allows code to be deployed with high confidence.

Exercise 4:

Configure the web application and Version control using Git using Git commands and version control operations
Create the web application repository, add all the team members & enforce some branch and review protections.

Linux Operating System

Linux is an open-source operating system. It is like Windows, Mac, Android, etc.

It is an commercial OS. It consists of three parts: Kernal, Shell and Programs. Linux includes all topics of Linux OS such as Linux commands, Directories, Files, Man Pages, File Contents, File Permissions, shells, VI editor etc. Father of Linux is *Linus Torvalds* 17 September 1991.

Linux User Interface

User interface

The user interface is called a shell. It is either a GUI (graphical user interface), a CLI (command-line interface), or controls attached to the related hardware, which is normal for embedded systems. The default user interface is graphical for desktop systems. However, the CLI is available by terminal emulator windows or on an isolated virtual console. Command-line interface shells are text-based UIs, which utilize text for both output and input.

Linux Features

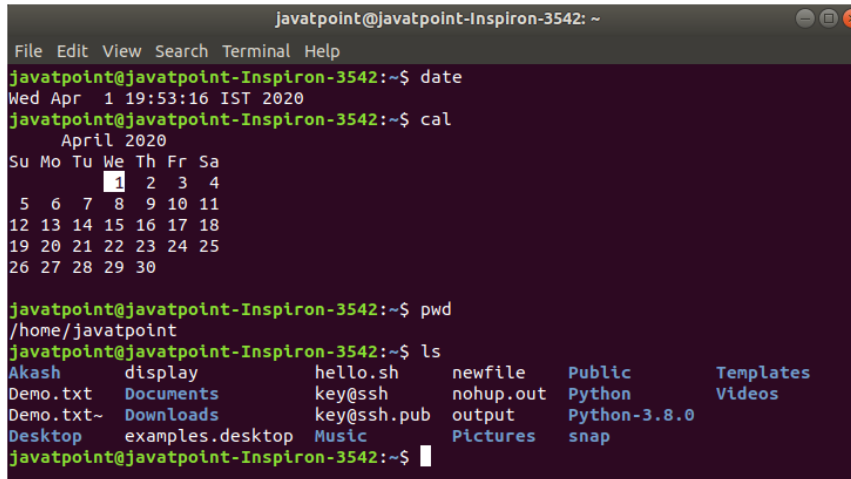
- **Multiuser capability:** Multiple users can access the same system resources like memory, hard disk, etc. But they have to use different terminals to operate.
- **Multitasking:** More than one function can be performed simultaneously by dividing the CPU time intelligently.
- **Portability:** Portability doesn't mean it is smaller in file size or can be carried in pen drives or memory cards. It means that it support different types of hardware.
- **Security:** It provides security in three ways namely authenticating (by assigning password and login ID), authorization (by assigning permission to read, write and execute) and encryption (converts file into an unreadable format).
- **Graphical User Interface (X Window system):** Linux is command line based OS but it can be converted to GUI based by installing packages.
- **Support's customized keyboard:** As it is used worldwide, hence supports different languages keyboards.
- **Application support:** It has its own software repository from where users can download and install many applications.
- **File System:** Provides hierarchical file system in which files and directories are arranged.
- **Open Source:** Linux code is freely available to all and is a community based development project.

What is Shell

If we are a new Linux user, and we open the terminal, it is assumed that we are well confused as to what to do with it. Here the Shell comes in the role.

The terminal contains the shell; it allows us to execute the commands to interact with the system. We can perform various operations such as store and retrieve data, process information, and various other simple as well as complex tasks.

To open the terminal, press **CTRL+ALT+T** keys. Perform some basic operations such as **date**, **cal**, **ls**, and **pwd** to take a tour with it.



```
javatpoint@javatpoint-Inspiron-3542: ~  
File Edit View Search Terminal Help  
javatpoint@javatpoint-Inspiron-3542:~$ date  
Wed Apr  1 19:53:16 IST 2020  
javatpoint@javatpoint-Inspiron-3542:~$ cal  
April 2020  
Su Mo Tu We Th Fr Sa  
          1  2  3  4  
 5  6  7  8  9 10 11  
12 13 14 15 16 17 18  
19 20 21 22 23 24 25  
26 27 28 29 30  
javatpoint@javatpoint-Inspiron-3542:~$ pwd  
/home/javatpoint  
javatpoint@javatpoint-Inspiron-3542:~$ ls  
Akash      display      hello.sh     newfile      Public       Templates  
Demo.txt   Documents    key@ssh      nohup.out    Python       Videos  
Demo.txt~  Downloads    key@ssh.pub  output       Python-3.8.0  
Desktop    examples.desktop Music         Pictures     snap  
javatpoint@javatpoint-Inspiron-3542:~$
```

Linux Commands

Linux Directory Commands

1. pwd Command

The pwd command is used to display the location of the current working directory.

Syntax:

\$ pwd

2. mkdir Command

The mkdir command is used to create a new directory under any directory.

Syntax:

\$ mkdir <directory name>

3. rmdir Command

The rmdir command is used to delete a directory.

Syntax:

\$ rmdir <directory name>

4. ls Command

The ls command is used to display a list of content of a directory.

Syntax:

\$ ls

5. cd Command

The cd command is used to change the current directory.

Syntax:

\$ cd <directory name>

Linux File commands

6. touch Command

The touch command is used to create empty files. We can create multiple empty files by executing it once.

Syntax:

\$ touch <file name>

\$ touch <file1> <file2> ...

7. cat Command

The cat command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.

Syntax:

\$ cat [OPTION]... [FILE]..

To create a file

Eg:

cat > <file name>

// Enter file content

Press "**CTRL+ D**" keys to save the file. To display the content of the file.

8. rm Command

The rm command is used to remove a file.

Syntax:

\$ rm <file name>

9. cp Command

The cp command is used to copy a file or directory.

Syntax:

To copy in the same directory:

\$ cp <existing file name> <new file name>

10. mv Command

The mv command is used to move a file or a directory from one location to another location.

Syntax:

\$ mv <file name> <directory path>

11. rename Command

The rename command is used to rename files. It is useful for renaming a large group of files.

Syntax:

\$ rename 's/old-name/new-name/' files

For example, to convert all the text files into pdf files, execute the below command:

\$ rename 's/\.txt\$/\.pdf/' *.txt

Linux File Content Commands

12. head Command

The head command is used to display the content of a file. It displays the first 10 lines of a file.

Syntax:

\$ head <file name>

13. tail Command

The tail command is similar to the head command. The difference between both commands is that it displays the last ten lines of the file content. It is useful for reading the error message.

Syntax:

\$ tail <file name>

14. tac Command

The tac command is the reverse of cat command, as its name specified. It displays the file content in reverse order (from the last line).

Syntax:

\$ tac <file name>

15. more command

The more command is quite similar to the cat command, as it is used to display the file content in the same way that the cat command does. The only difference between both commands is that, in case of larger files, the more command displays screenful output at a time.

In more command, the following keys are used to scroll the page:

ENTER key: To scroll down page by line.

Space bar: To move to the next page.

b key: To move to the previous page.

/ key: To search the string.

Syntax:

\$ more <file name>

16. less Command

The less command is similar to the more command. It also includes some extra features such as 'adjustment in width and height of the terminal.' Comparatively, the more command cuts the output in the width of the terminal.

Syntax:

\$ less <file name>

Linux User Commands

17. su Command

The su command provides administrative access to another user. In other words, it allows access of the Linux shell to another user.

Syntax:

\$ su <user name>

18. id Command

The id command is used to display the user ID (UID) and group ID (GID).

Syntax:

\$ id

19. useradd Command

The useradd command is used to add or remove a user on a Linux server.

Syntax:

useradd username

20. passwd Command

The passwd command is used to create and change the password for a user.

Syntax:

\$ passwd <username>

21. groupadd Command

The groupadd command is used to create a user group.

Syntax:

\$ groupadd <group name>

22. grep Command

The grep is the most powerful and used filter in a Linux system. The 'grep' stands for "**global regular expression print.**" It is useful for searching the content from a file. Generally, it is used with the pipe.

Syntax:

\$ command | grep <searchWord>

Eg:

\$ cat marks.txt | grep 89

23. comm Command

The 'comm' command is used to compare two files or streams. By default, it displays three columns, first displays non-matching items of the first file, second indicates the non-matching item of the second file, and the third column displays the matching items of both files.

Syntax:

\$ comm <file1> <file2>

24. sed command

The sed command is also known as **stream editor**. It is used to edit files using a regular expression. It does not permanently edit files; instead, the edited content remains only on display. It does not affect the actual file.

Syntax:

\$ command | sed 's/<oldWord>/<newWord>/'

25. gzip Command

The gzip command is used to truncate the file size. It is a compressing tool. It replaces the original file by the compressed file having '.gz' extension.

Syntax:

\$ gzip <file1> <file2> <file3>...

26. gunzip Command

The gunzip command is used to decompress a file. It is a reverse operation of gzip command.

Syntax:

\$ gunzip <file1> <file2> <file3>. .

27. zcat Command

The zcat command is used to display the compressed files.

Syntax:

\$ zcat <file name>

28. clear Command

Linux **clear** command is used to clear the terminal screen.

Syntax:

\$ clear

Linux Networking Commands

29. ip Command

Linux **ip** command is an updated version of the ipconfig command. It is used to assign an IP address, initialize an interface, disable an interface.

Syntax:

\$ ip a or \$ ip addr

30. ssh Command

Linux **ssh** command is used to create a remote connection through the ssh protocol.

Syntax:

\$ ssh user_name@host(IP/Domain_name)</p>

31. mail Command

The mail command is used to send emails from the command line.

Syntax:

\$ mail -s "Subject" <recipient address>

32. host Command

The host command is used to display the IP address for a given domain name and vice versa. It performs the DNS lookups for the DNS Query.

Syntax:

\$ host <domain name> or <ip address>

33. whoami

It tells you about the system's username

Syntax:

\$ whoami

34. who

The who command gives the information about the users logged on to the system.

Syntax:

\$ who

35. who am i

This command displays the information about the current user only.

Syntax:

\$ w

This command tells about the users who are logged in and what are they doing.

36.who am i

Syntax:

\$ w

37. id

This command tells about your user id, primary group id, and a list of groups that belongs to you.

Syntax:

`$ id`

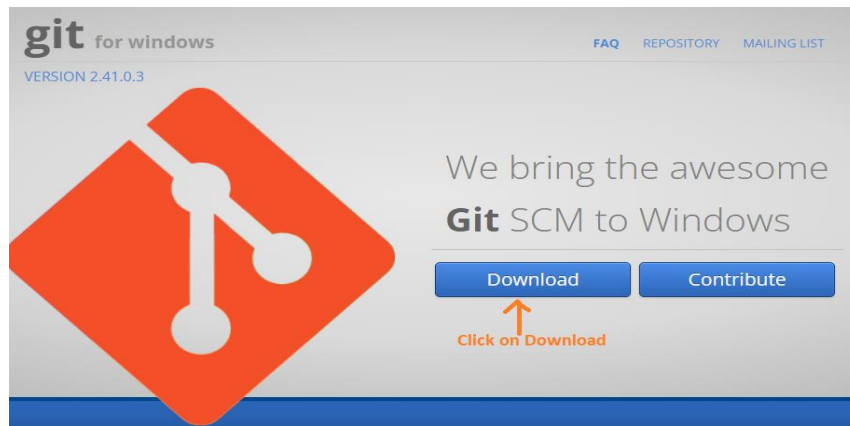
Vi shortcuts

The best way to learn Vi is to create a new file and try it out for yourself. Feel free to use the common keyboard shortcut list below to help you learn Vi's extensive vocabulary. This list of shortcuts is by no means exhaustive, but they will enable you to edit files and learn Vi in a short amount of time.


- `$ vi <filename>` — Open or edit a file.
- `i` — Switch to Insert mode.
- **Esc** — Switch to Command mode.
- `:w` — Save and continue editing.
- `:wq` or `ZZ` — Save and quit/exit vi.
- `:q!` — Quit vi and do not save changes.
- `yy` — Yank (copy) a line of text.
- `p` — Paste a line of yanked text below the current line.
- `o` — Open a new line under the current line.
- `O` — Open a new line above the current line.
- `A` — Append to the end of the line.
- `a` — Append after the cursor's current position.
- `I` — Insert text at the beginning of the current line.
- `b` — Go to the beginning of the word.
- `e` — Go to the end of the word.
- `x` — Delete a single character.
- `dd` — Delete an entire line.
- `Xdd` — Delete X number of lines.
- `Xyy` — Yank X number of lines.
- `G` — Go to the last line in a file.
- `XG` — Go to line X in a file.
- `gg` — Go to the first line in a file.
- `:num` — Display the current line's line number.
- `h` — Move left one character.
- `j` — Move down one line.
- `k` — Move up one line.
- `l` — Move right one character.

Git Installation on Windows

Step 1: Download the latest version of Git and choose the 64/32 bit version. After the file is downloaded, install it in the system. <https://gitforwindows.org/>



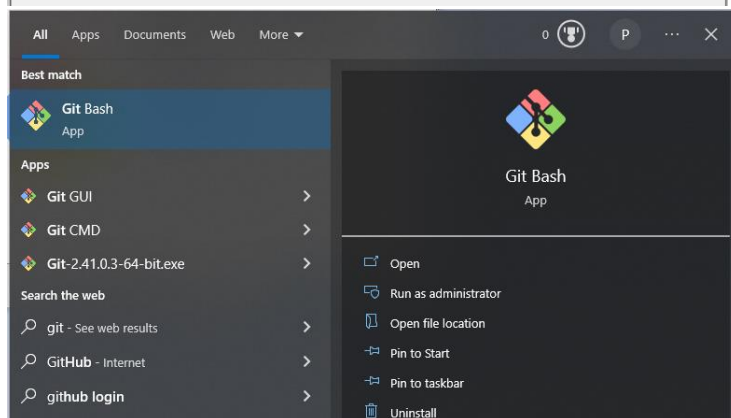
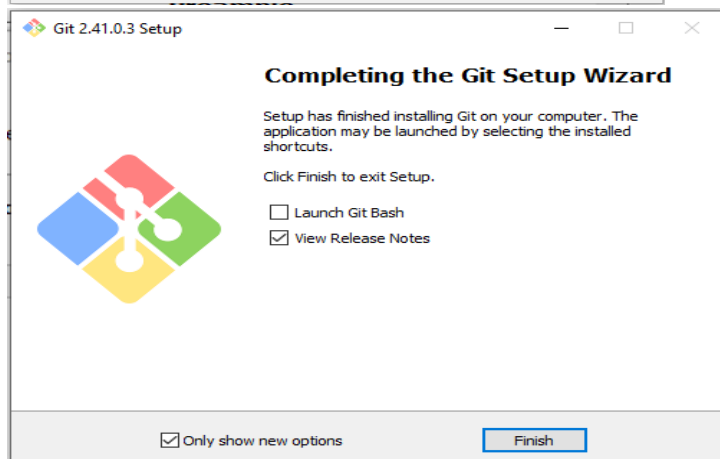
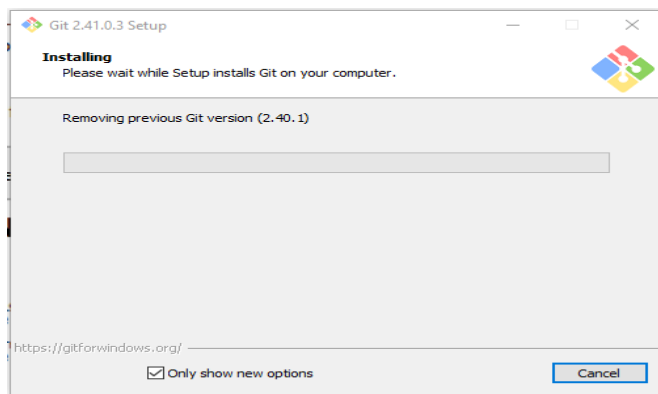
Recent Downloads

 Git-2.41.0.3-64-bit.exe
↓ 6.6/58.4 MB • Opening in 2 minutes...

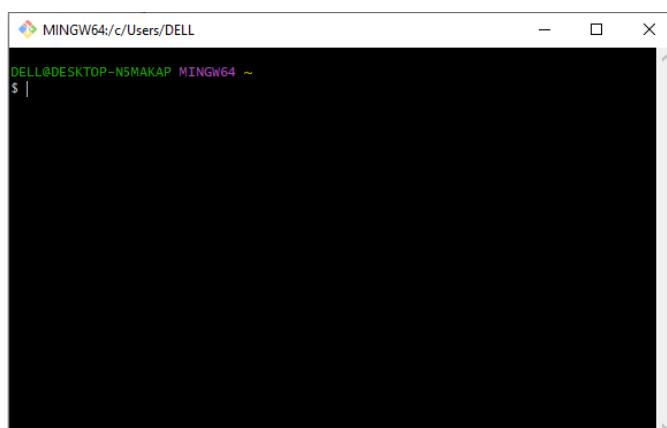
Go to Downloads then you will find Git.exe file Download

After Download click on open





click on Open.



now you can execute Git Commands.

- **Check the Git version:**
\$ git --version
- **For any help, use the following command:**
\$ git help config
This command will lead you to a browser of config commands. Basically, the help the command provides a manual from the help page for the command just following it (here, it's config).
- **Another way to use the same command is as follows:**
\$ git config --help
- **Create a local directory using the following command:**
\$ mkdir test
\$ cd test
- **Initialize the directory:**
\$ git init
- **Enter the Git bash interface and type in the following command to check the status:**
\$ git status
- **Add the "demo" to the current directory using the following command:**
\$ git add demo.txt
- **make a commit using the following command:**
\$ git commit -m "committing a text file"
- **Link the Git to a Github Account:**
\$ git config --global user.username
- **Note:** simplilearn-github is the username on the Github account.
- **Git bash and link the remote and local repository using the following command:**
\$ git remote add origin <link>
- **Push the local file onto the remote repository using the following command:**
\$ git push origin master
- Move back to Github and click on "test_demo" and check if the local file "demo.txt" is pushed to this repository.

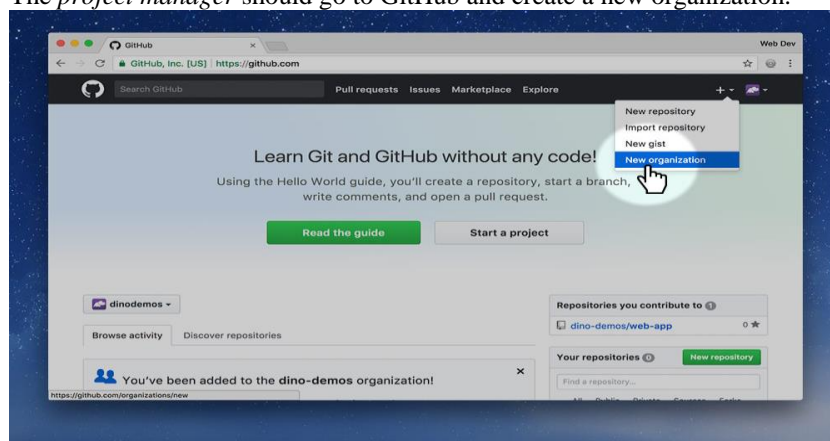
Goal

We're going to start the basic repository, add all our teammates & set up a better GitHub code review process to enforce code reviews for each other.

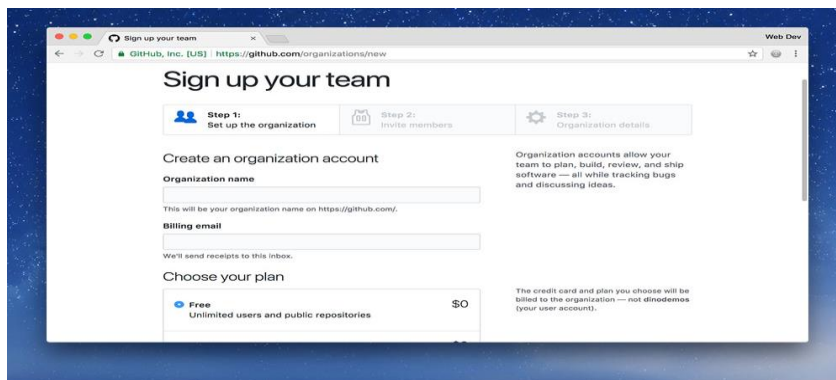
1. Create an organization

For this project you'll all be owners of the code-base and the application. To better support this we're going to create an organization, that way the repository won't live directly in a single user's GitHub account.

The *project manager* should go to GitHub and create a new organization.

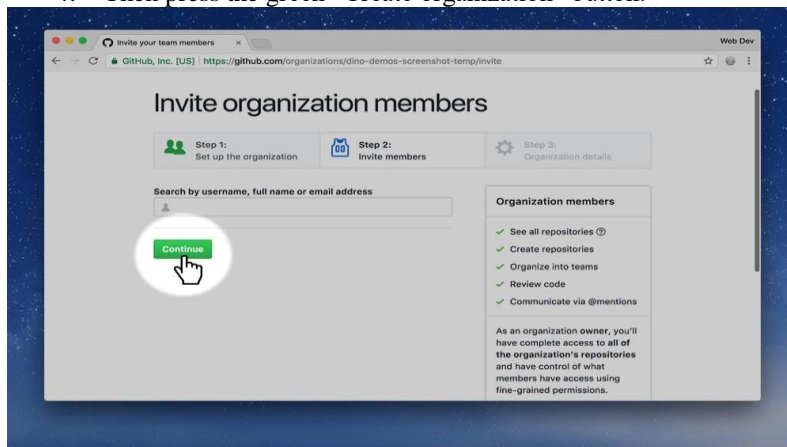


In the + menu go to "New organization".

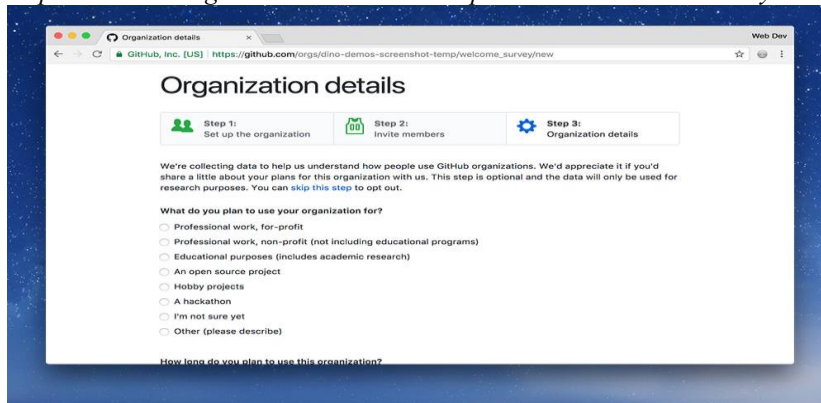


Fill in following pieces of information about your organization:

1. The organization URL, will end up like this: <https://github.com/your-org-name>
2. The email address of your project manager.
3. Select the “Free” organization type.
4. Then press the green “Create organization” button.



Skip the “Invite organization members” step and continue to the survey.



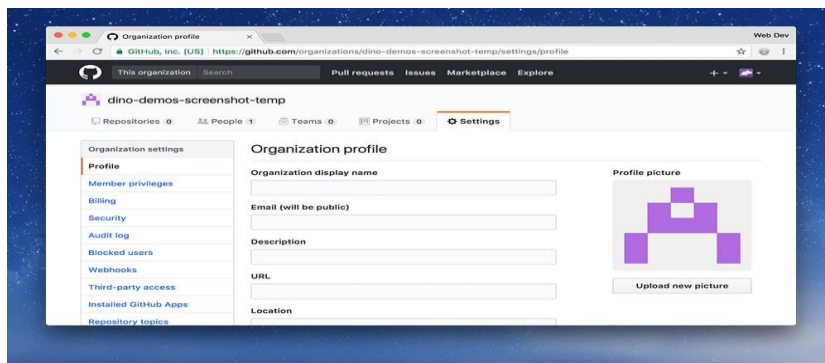
On the survey, use the following answers to the three questions:

1. “Educational purposes (includes academic research)”
2. “A few weeks to months”
3. “5 or fewer”

Then press “Submit”.

Customize the organization

If you'd like you can also customize some properties of the organization by going to the “Settings” tab.

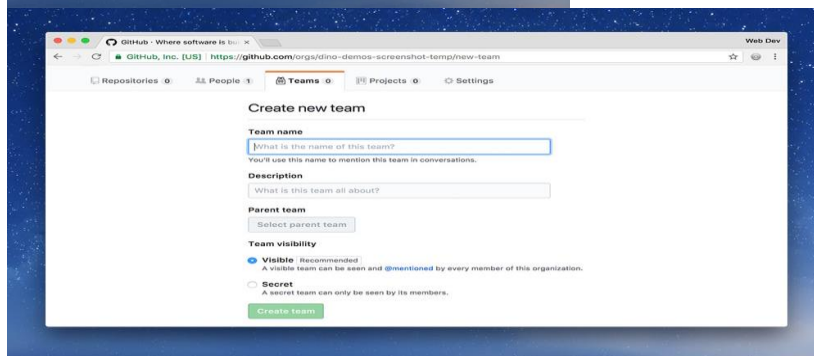
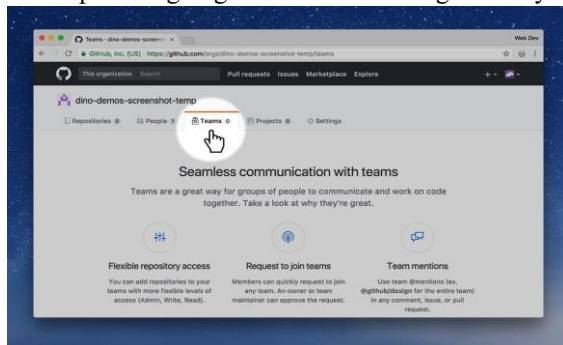


Upload a profile picture of your organization and enter a better “Organization display name”. Fill out any other details you’d like.

2. Add all the team members

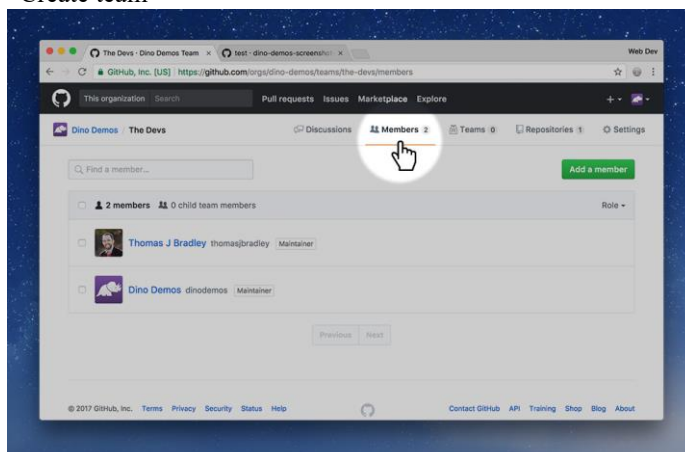
Next up we’re going to create a team to give everybody equal permissions to the organization and its repositories.

Click the “Teams” tab and press “New Team”.

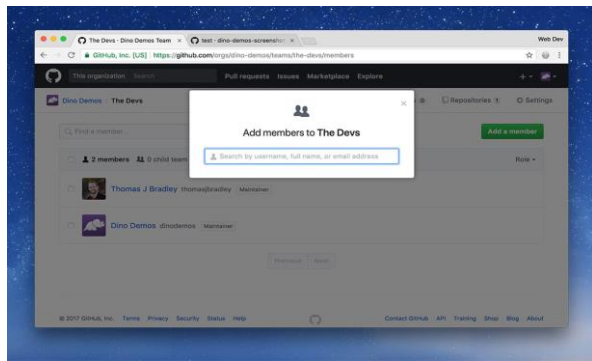


Fill in all the details about your team: the name you came up with, a description of your awesomeness, then press “Create team”

Next up we need to add members to the team because



the project manager is the only member right now. So go to the “Members” link.



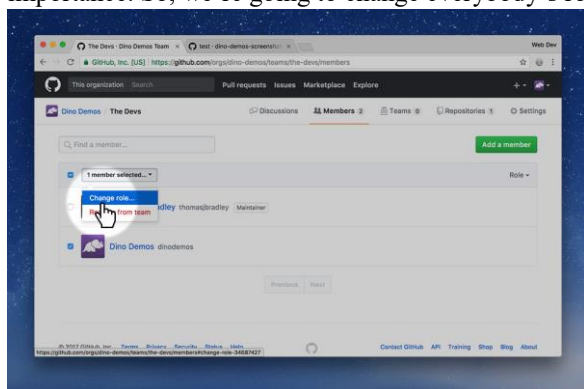
For each person in your group press “Add member” and type in their GitHub username. It will autocomplete.

3. Approve being a team member

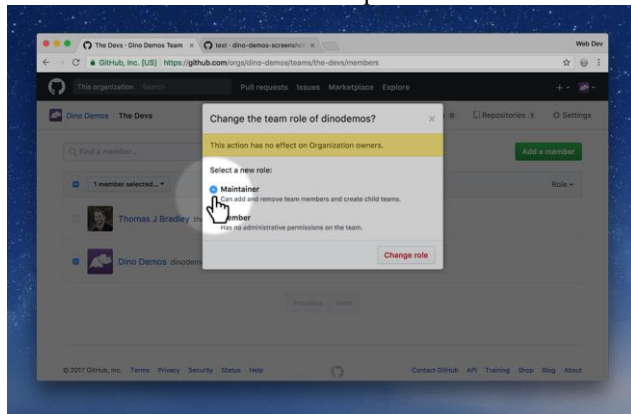
Now every person in the group will get an email from GitHub. **You’ll each have to approve being added to the team before continuing to the next step.**

4. Make everyone a maintainer

Right now the project manager is the most important member of the team but we want everybody to have equal importance. So, we’re going to change everybody’s role to be “Maintainer”.



Select all the team members except the PM and click the “No members selected” button. Then press “Change role”.

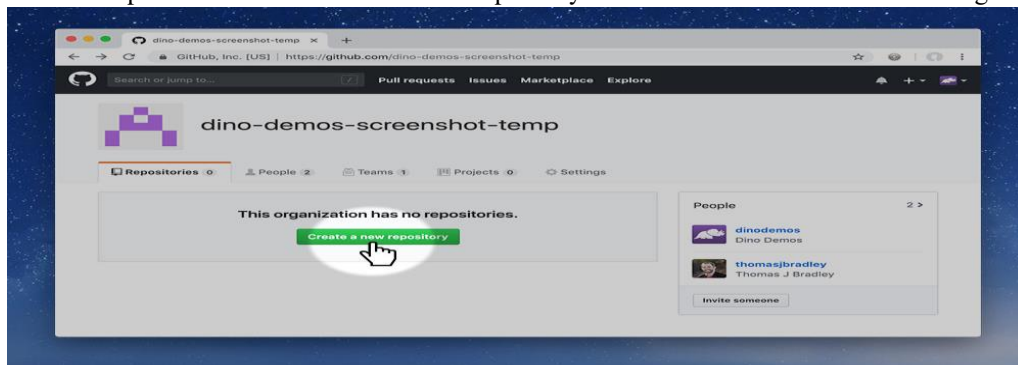


Change everybody to the “Maintainer” role and save the changes.

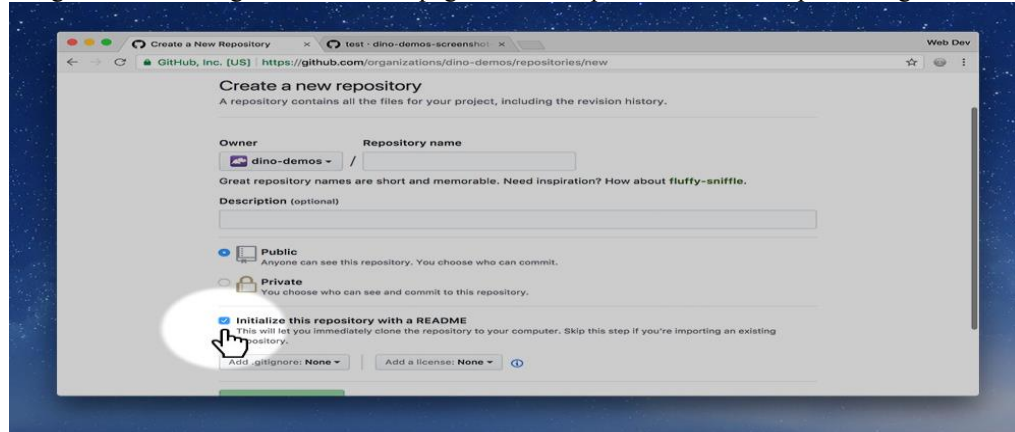
Now everybody can contribute to the project on the same level including configuring settings and managing teams.

5. Create the repo in the org

The whole point of this is to make a shared repository to write code in—that’s what we’re going to do next.



So, go back to the organization's start page, to the "Repositories" tab and press the green "New" button.



Fill in the standard details for your repository, you could name the repo something generic, like `web-app`, because you haven't decided what you're actually making yet.

Make sure to enable "Initialize this repository with a README!"

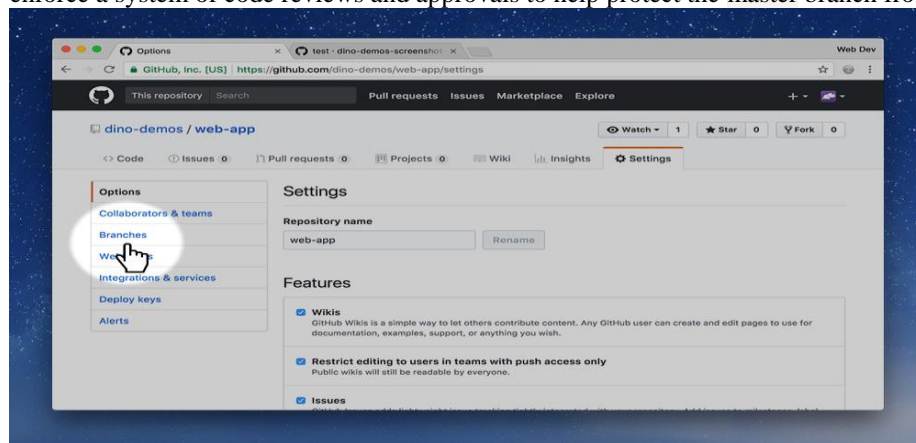
6. Protect the master branch

As part of good software development practices we want to isolate the development code from the good, live code.

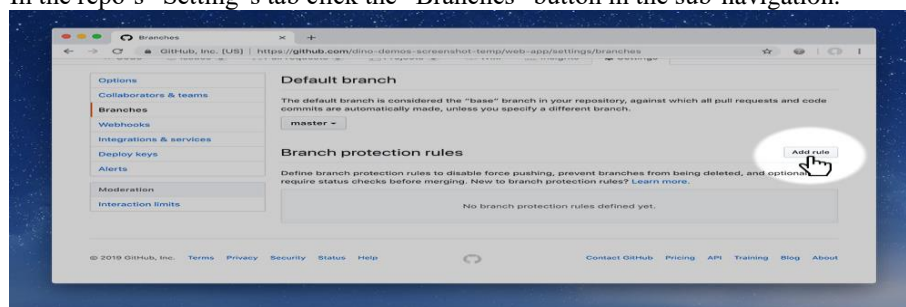
Our master branch should always be in a *good* state so that it can be live on the website.

We do all our work on other branches and merge those into master only when the code is good enough. This process is a little different from what we've been following before.

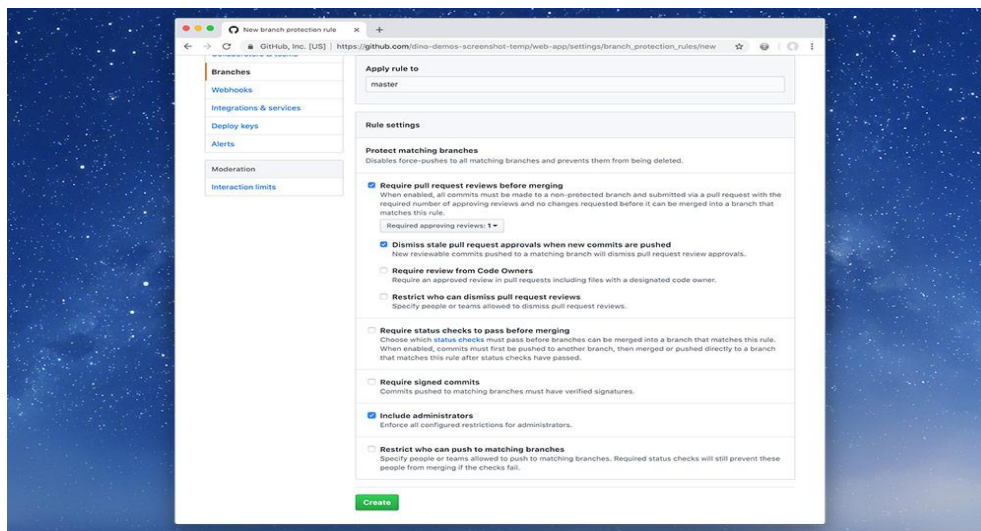
So we're going to protect the master branch so we can't accidentally commit or push into that branch. We're going to enforce a system of code reviews and approvals to help protect the master branch from getting ugly.



In the repo's "Setting's" tab click the "Branches" button in the sub-navigation.



A little bit down the page is a "Branch protection rules" repository. Press the "Add rule" button.



Finally we need to configure how the master branch will be protected from our messing about.

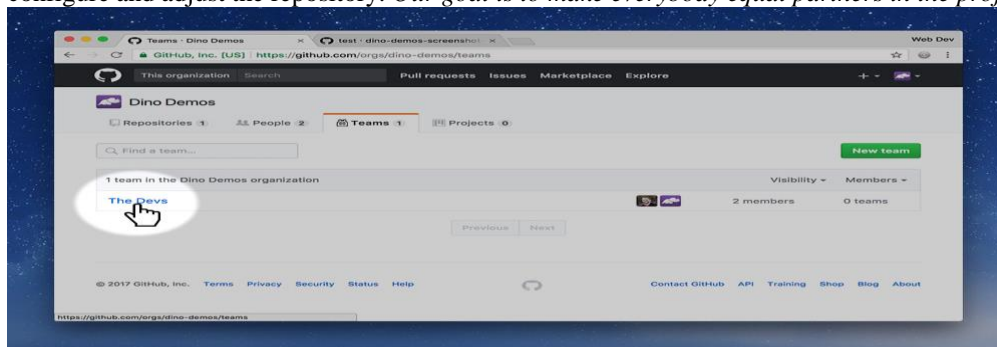
Select the following options on the screen and save:

1. Type master into the “*Apply rule to*” input field
This will prevent you from committing and pushing to the master branch—you’ll have to create a different branch.
2. “*Require pull request reviews before merging*”
This will require everybody to get someone else to approve their code changes before they can be implemented into the master branch.
3. “*Dismiss stale pull request approvals when new commits are pushed*”
If someone has approved the code changes, but then the original author creates a new commit, the approval will be voided.
4. “*Include administrators*”
Enforces these rules on everybody, even if they are an admin of the project (which you all are).

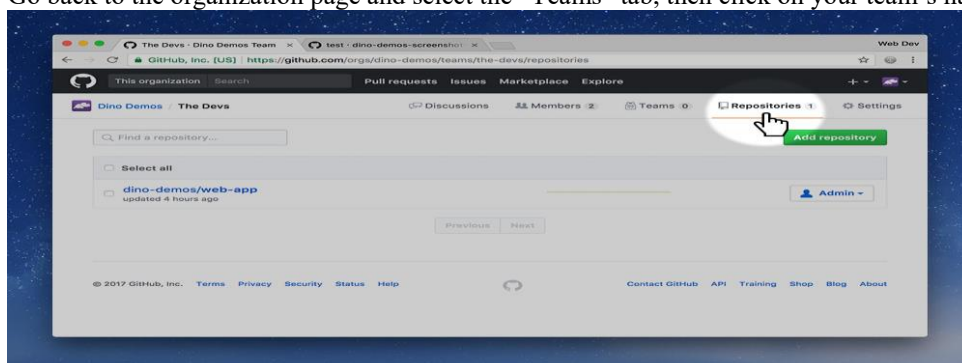
Make sure to save the changes.

7. Make the team members admins

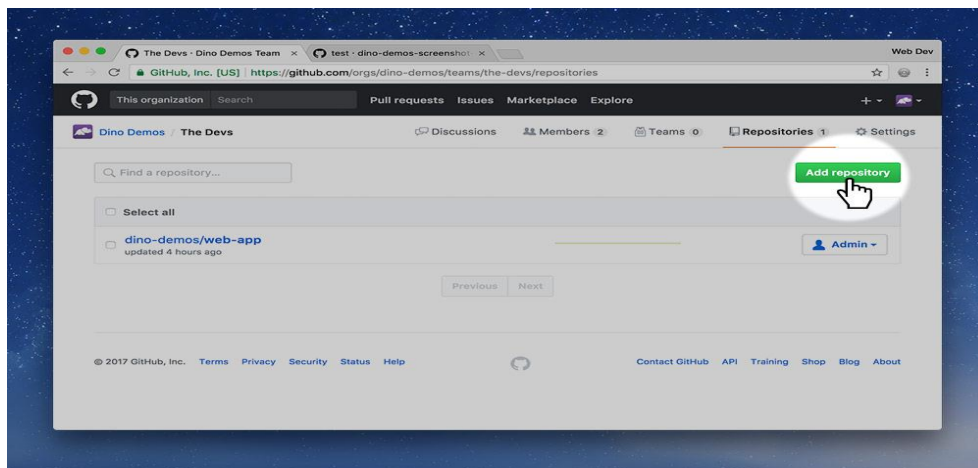
Next up we’re going to give everybody administrator access to this repository. This will allow everybody access to configure and adjust the repository. *Our goal is to make everybody equal partners in the project.*



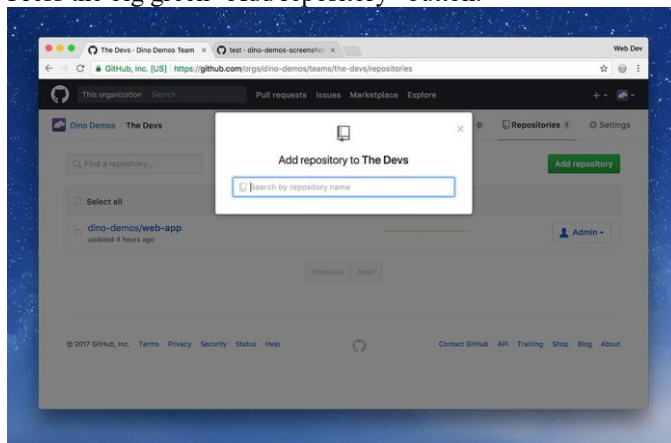
Go back to the organization page and select the “Teams” tab, then click on your team’s name.



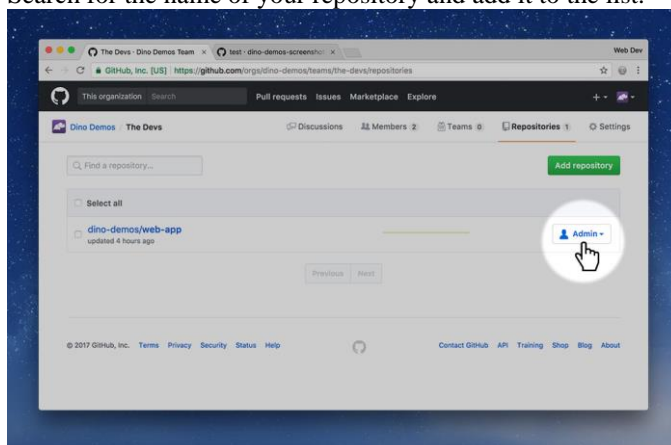
Go to the “Repositories” tab for your team.



Press the big green “Add repository” button.



Search for the name of your repository and add it to the list.



Double check that your team is set as “Admin” for the repository—which I think is the default.

Now our organization and team is ready to go!

8.Clone to everybody’s computer

Finally, something for all the team members to do, though it’s really small at this point: clone the repo to your computer.

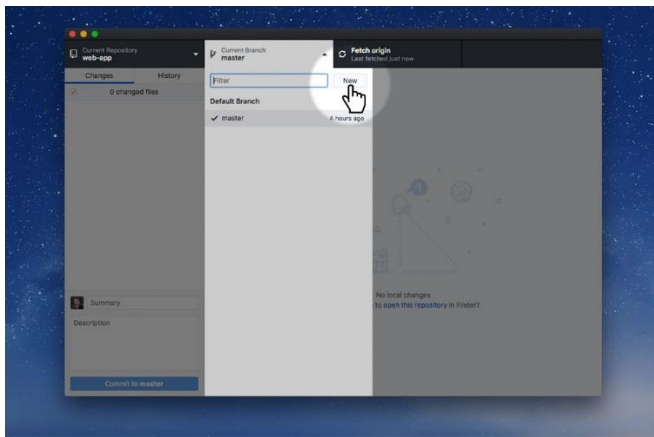
Everybody should go to the web-app repository and clone it to their computer using the standard procedure.

9.Create a temporary branch

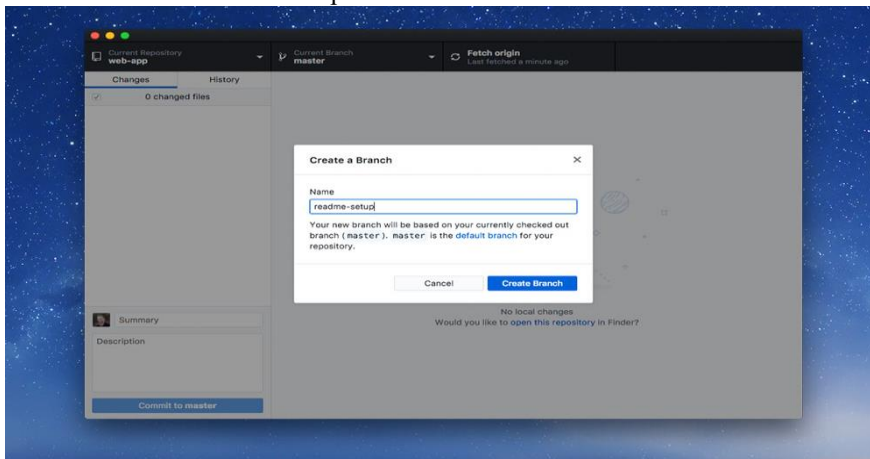
This step is for the dev lead to complete. We’re going to create the readme with some basic information for now that can be filled in with more detail later.

After cloning the repository we’re going to make a new branch in the GitHub Desktop app.

Remember that we cannot commit and push to the master branch.



Click the branches button and press the “New” button.



Name the new branch something like setup-readme then press “Create branch”. This is the process you’ll go through every time you code a new feature—we’ll discuss this in more detail later.

10. Add a descriptive readme

Open up the folder in your code editor and edit the `README.md` file, adding a few details. Something like this will work:

Web app prototype

This repository is a prototype for a web app created in Graphic Design’s Web Dev 6 course.

Team members

- Thomas J Bradley <<https://github.com/thomasjbradley>>
- Dino Demos <<https://github.com/dinodemos>>

Save and commit that. Make sure you’re not committing to the master branch.

1. Commit conventions

Don’t forget to follow the [Git commit best practices](#).

10. Add a GitIgnore file

Create a new file in the repository named `.gitignore` and add this information to it. **Copying is enabled for efficiency.**

```
# Jekyll
_site
.jekyll-metadata
.jekyll-cache

# Adobe files
*.pdf
*.ai
*.psd
*.indd
*.indb

# Compressed files
*.zip
*.gz
```

```

*.tar
*.7z

# Type faces
*.otf
*.ttf
*.woff
*.eot
*.ttc

# Video & audio
*.mov
*.mp4
*.m4v
*.f4v
*.f4p
*.ogv
*.webm
*.flv
*.mp3
*.m4a
*.f4a
*.f4b
*.oga
*.ogg
*.opus

# Folder view configuration files
.DS_Store
Desktop.ini

# Thumbnail cache files
._*
Thumbs.db

# Files that might appear on external disks
.Spotlight-V100
.Trashes

```

Save and commit that.

12. Add an EditorConfig file

Create a new file named `.editorconfig` in the folder and add this information to it. **Copying is enabled for efficiency.**

```

; editorconfig.org
root = true

```

```

[*]
charset = utf-8

```

```

indent_style = space
indent_size = 2

```

```

trim_trailing_whitespace = true
end_of_line = lf
insert_final_newline = true

```

Save and commit that.

13. Add a GitAttributes file

You only need perform this step if one or more of your team members use a Windows computer—but it's good practice to do it anyways.

Create another new file named `.gitattributes` and add this information to it. **Copying is enabled for efficiency.**

```

# Force Unix LF to make code consistent across platforms
# Helps Markbot cheat detection to work properly

```

```

*.html text eol=lf
*.css text eol=lf

```

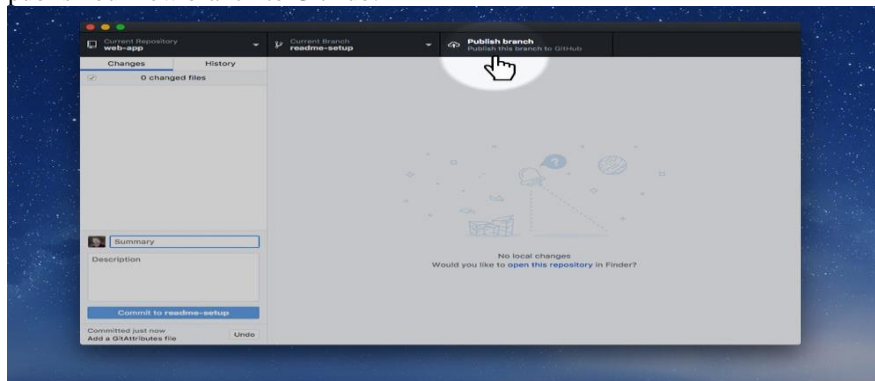
```
*.js text eol=lf
*.md text eol=lf
*.yml text eol=lf
*.txt text eol=lf
*.svg text eol=lf
*.lock text eol=lf
```

```
.editorconfig text eol=lf
```

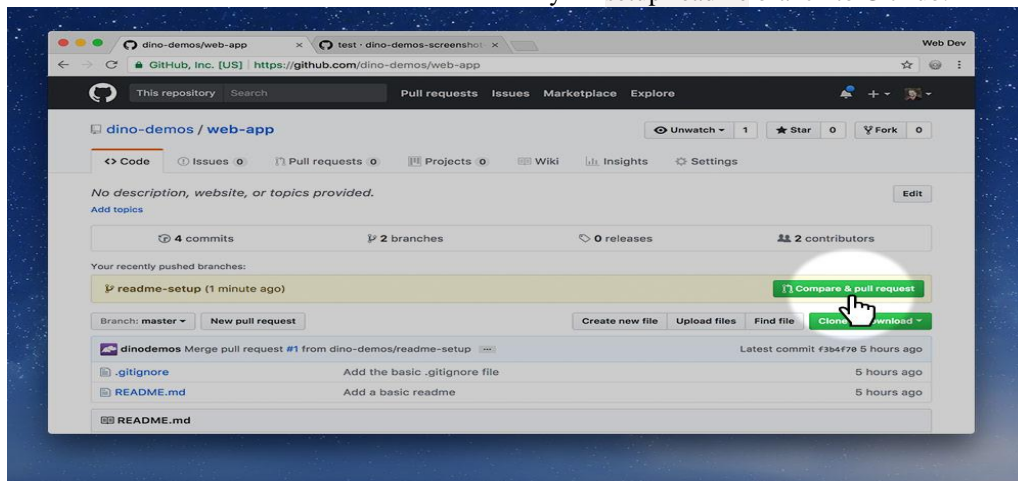
Save and commit that.

14. Create a pull request

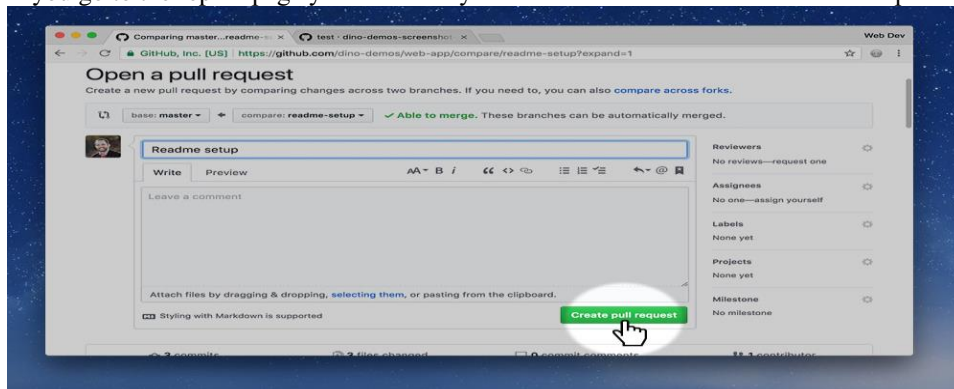
Our next goal is to get this code into the `master` branch so it's available on everybody's computer. Step one is to publish our new branch to GitHub.



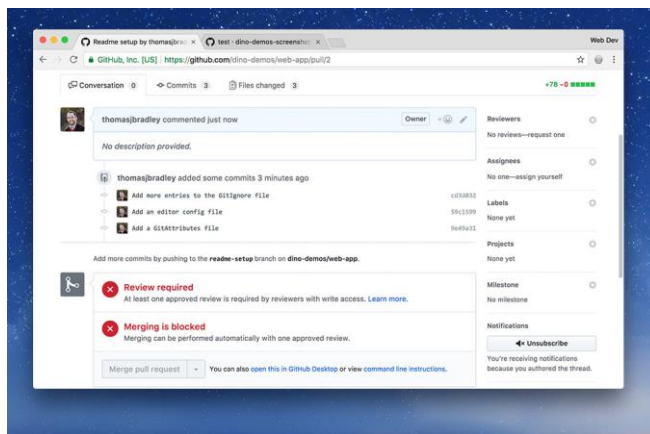
Press the “Publish branch” button which will send your setup-readme branch to GitHub.



If you go to the repo's page you'll see the yellow new branch alert. Press the “Compare & pull request” button.



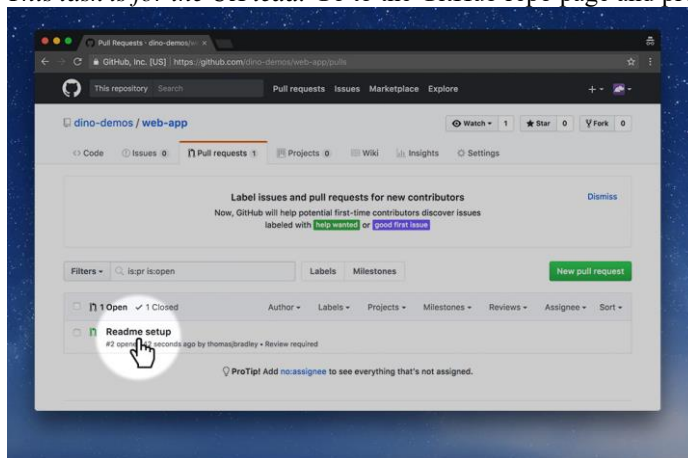
Describe what changes you made in the branch you just published and press the “Create pull request” button.



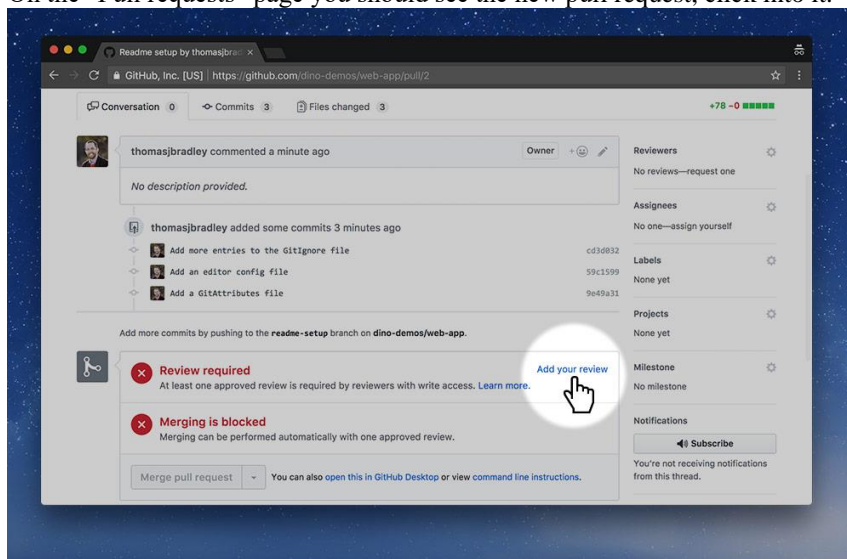
You'll notice that next screen says a code review is required. So, now we hand the process off to another teammate, the UX lead.

15. Approve & merge the request

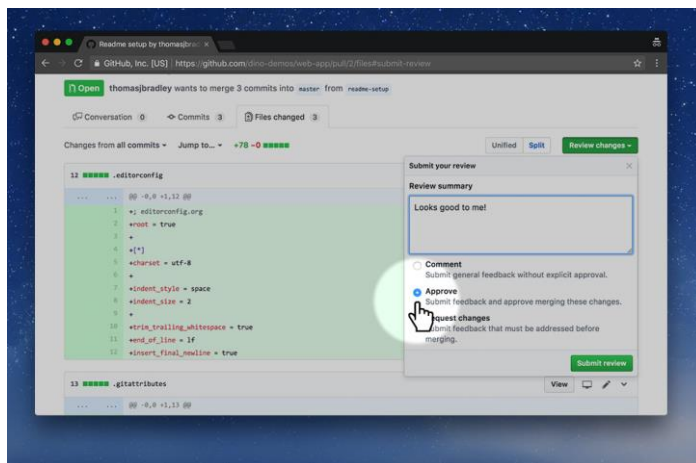
This task is for the UX lead. Go to the GitHub repo page and press the “Pull requests” tab.



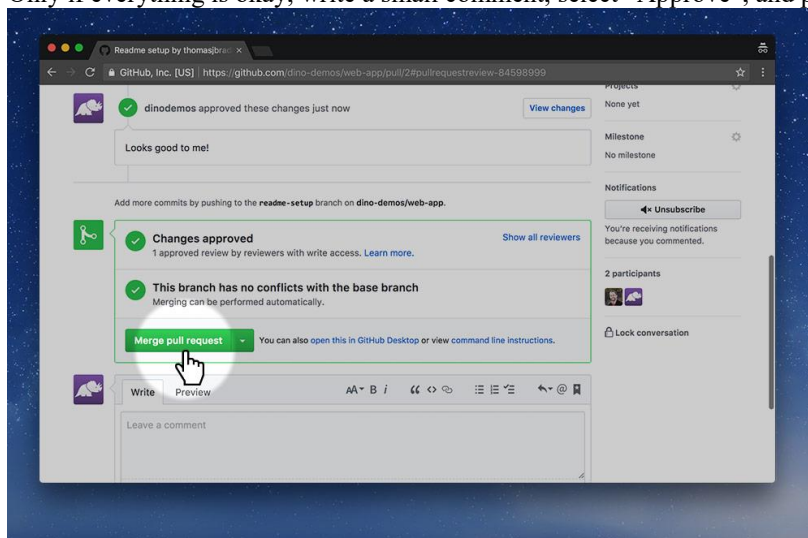
On the “Pull requests” page you should see the new pull request, click into it.



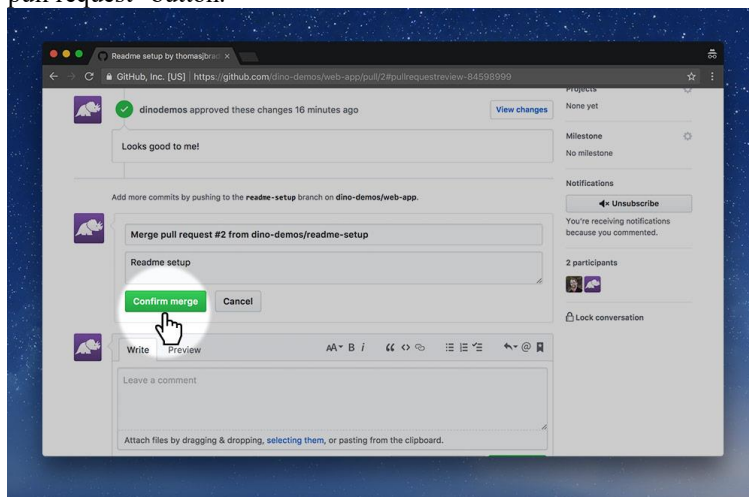
You'll notice that the PR requires a code review to move forward. It's up to you to do that review. Press the “Add your review” link.



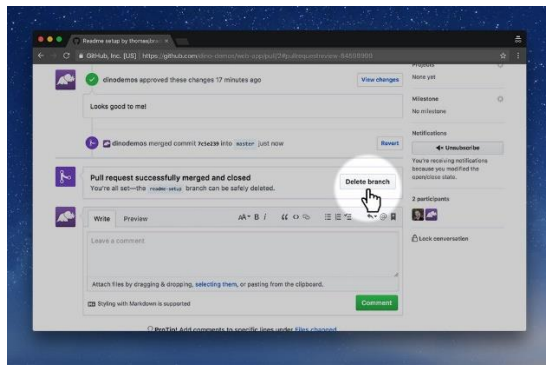
On the next screen you'll see all the code contributions that were made. Double-check there's no spelling mistakes or problems.
Only if everything is okay, write a small comment, select "Approve", and press the "Submit review" button.



Since the code has been approved, we're now ready to merge those changes into the `master` branch. Press the "Merge pull request" button.



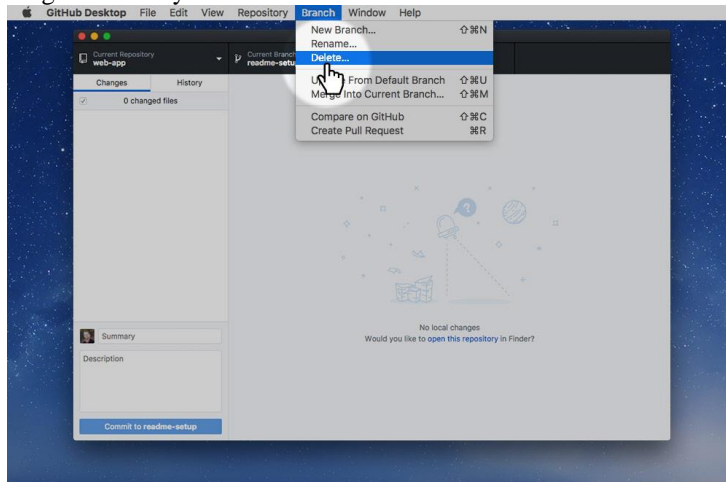
Then press "Confirm merge"



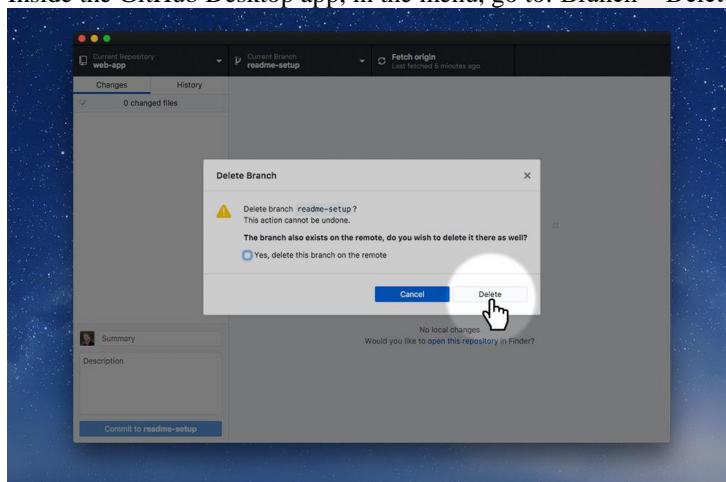
Now that the code is merged, we no longer need the branch so press the “Delete branch” button. *That only deletes it from GitHub.com, not from our local computer. We’re going to do that next.*

16.Delete the local branch

Back to the dev lead. Now that the code has been merged into the master branch the local setup-readme branch is no longer necessary so we need to delete it.



Inside the GitHub Desktop app, in the menu, go to: Branch > Delete... to remove the local branch.



Confirm the deletion by pressing the “Delete” button.

17.Pull the changes from GitHub

For everybody to complete. We want to make sure everybody has the most up-to-date version of the code on their computers, so they need to pull the most recent changes from GitHub’s website.

On your own computer go into the GitHub Desktop app and press the “Fetch origin” button. *If it switches and says “Pull” do that too.*

Double check that your local copy of the website has the correct README.md, .editorconfig, .gitignore and .gitattributes files.