TechClass

Mastering Version Control with Git

Introduction part (A)

1st edition

Lecturer: Farhad Eftekhari

Copyright Declaration

Content of this course have been provided from the teacher of this course (Farhad Eftekhari) personal experiences, using Atlassian Git content (under a Creative Commons Attribution 2.5 Australia License) and several mentioned online sources throughout the material of the course.

Course Links

Course www.techclass.co/courses/git

Slides www.techclass.co/courses/git/slides

Tutorial Videos www.techclass.co/courses/git/videos

Assignments www.techclass.co/courses/git/assignments

Files www.techclass.co/courses/git/files

Note: You need to be accepted in the course to have access to the course material.

Typographical conventions

 **Further study**	At the end of the slides, there will be a link for further study.
 **Homework**	There will be a question concerning the topic in the assignments.
 **Discussion**	A class/pair discussion concerning the topic.
 **Do it yourself**	Do the task yourself to have a hand-on experience.
 **Practical point**	A practical point to consider.
 **Back in history**	Comparison to the methods used to be before.
 **Be careful**	An important issue to consider.
 **Fun to learn**	The topic will not be included in the assignments and the quiz.

Difficulty Level Indicator

Difficulty Level Indicator will be placed on top of the slides.

Easy

Introduction to the material, basic concepts and their purposes.

Moderate

More practical material.

Hard

Advanced and complicated material (mostly the topics are for the students to be familiar with, and not completly be able to use them).

Slides Framework

Difficulty indicator

6/55

code

Title

Slide number

Tutorial video code
www.techclass.co/go/code

Content

Course name

Section name

Icons

Content

- What is Version Control?
- Benefits of Version Control
- Continuous Integration (CI)
- What is Git?
- How does Git work?
- Who needs Git?
  - Git for Developers
  - Git for Marketing
  - Git for Product Management
  - Git for Designers
  - Git for Customer Support
  - Git for anyone managing a budget
- Installing Git (Windows-Mac-Linux)
- Git repository hosting services
- Introduction to GitHub

Our approach in this course

Concepts

- Version Control
- Continuous Integration
- Methods
- Git

Make it ready

- Users of Git
- How to install?
- Git hosting services
- GitHub

Start

- Setting up a repository
- Saving changes
- Inspecting a repository

Time Machine

- Viewing old commits
- Undoing Changes
- Rewriting history

Cooperation

- Syncing
- Making a Pull Request
- Using Branches

Definitions

VCS

- Stands for Version Control System

SCM

- Stands for Source Code Management

RCS

- Stands for Revision Control System

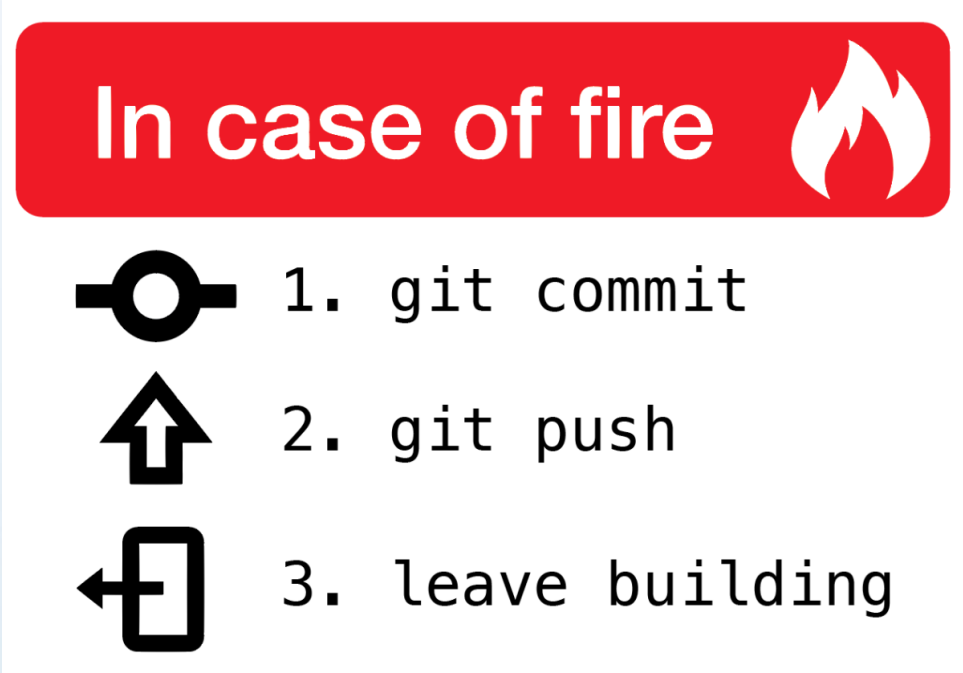
CI

- Stands for Continuous Integration

IDE

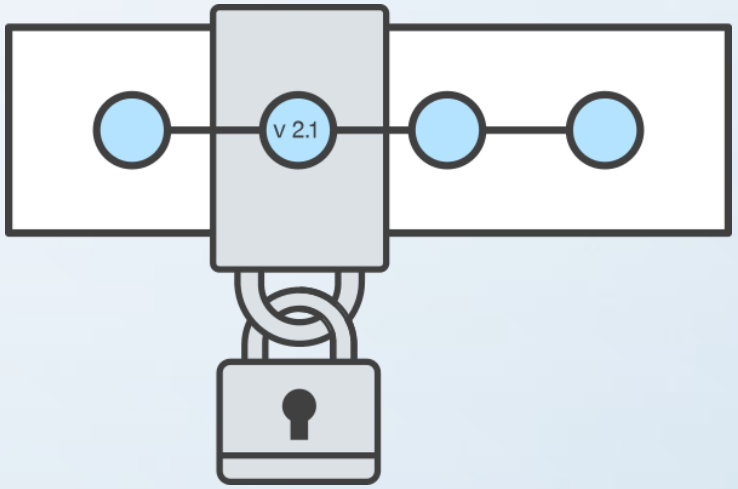
- Stands for Integrated Development Environments

Get it?!



After passing this course, you will get the joke, and find it funny! ☺

What is Version Control? (1/6)



- Tracking Changes
- Teamwork Collaboration
- Preventing Conflicts
- Faster Development
- Essential

What is Version Control? (2/6)

Version control systems are a category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

What is Version Control? (3/6)

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

What is Version Control? (4/6)

Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can‘t be trusted until it’s tested. So testing and development proceed together until a new version is ready.

What is Version Control? (5/6)

Good version control software supports a developer's preferred workflow without imposing one particular way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

What is Version Control? (6/6)

Version control software is an essential part of the every-day of the modern software team‘s professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn’t consider working without it even for non-software projects.

Benefits of Version Control (1/4)

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control are as follows.

Benefits of Version Control (2/4)

A complete long-term change history of every file

- This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.

Benefits of Version Control (3/4)

Branching and merging

- Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

Benefits of Version Control (4/4)

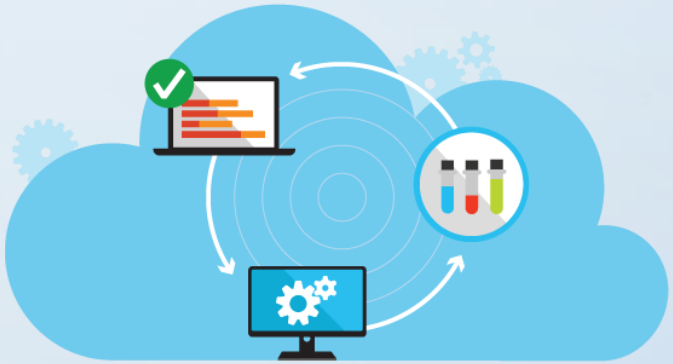
Traceability

- Being able to trace each change made to the software and connect it to project management and bug tracking software such as JIRA, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

Continuous Integration (CI) (1/5)

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

By integrating regularly, you can detect errors quickly, and locate them more easily. (Source: ThoughWorks)



Continuous Integration (CI) - Benefits (2/5)

- Say goodbye to long and tense integrations
- Increase visibility which enables greater communication
- Catch issues fast and nip them in the bud
- Spend less time debugging and more time adding features
- Proceed in the confidence you're building on a solid foundation
- Stop waiting to find out if your code's going to work
- Reduce integration problems allowing you to deliver software more rapidly

Continuous Integration (CI) - Practices (3/5)

- Maintain a single source repository
- Automate the build
- Make your build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Continuous Integration (CI) – How to do (4/5)

- Developers check out code into their private workspaces.
- When done, commit the changes to the repository.
- The CI server monitors the repository and checks out changes when they occur.
- The CI server builds the system and runs unit and integration tests.
- The CI server releases deployable artefacts for testing.
- The CI server assigns a build label to the version of the code it just built.
- The CI server informs the team of the successful build.
- If the build or tests fail, the CI server alerts the team.
- The team fix the issue at the earliest opportunity.
- Continue to continually integrate and test throughout the project.

Continuous Integration (CI) – Responsibilities (5/5)

- Check in frequently
- Don't check in broken code
- Don't check in untested code
- Don't check in when the build is broken
- Don't go home after checking in until the system builds

What is Git? (1/5)

the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).

Linux

What is Git? (2/5)

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

- Performance
- Security
- Flexibility

What is Git? – Performance (3/5)

The raw performance characteristics of Git are very strong when compared to many alternatives. Committing new changes, branching, merging and comparing past versions are all optimized for performance. The algorithms implemented inside Git take advantage of deep knowledge about common attributes of real source code file trees, how they are usually modified over time and what the access patterns are.

What is Git? – Security (4/5)

Git has been designed with the integrity of managed source code as a top priority. The content of the files as well as the true relationships between files and directories, versions, tags and commits, all of these objects in the Git repository are secured with a cryptographically secure hashing algorithm called SHA1. This protects the code and the change history against both accidental and malicious change and ensures that the history is fully traceable.

What is Git? – Flexibility (5/5)

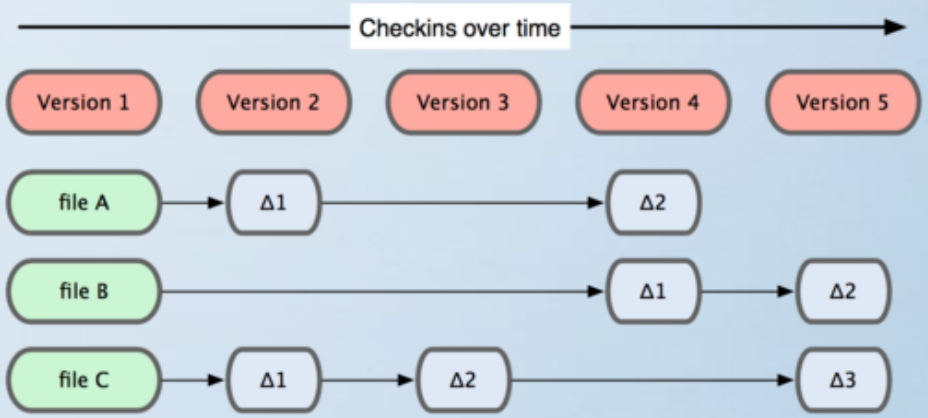
One of Git's key design objectives is flexibility. Git is flexible in several respects: in support for various kinds of nonlinear development workflows, in its efficiency in both small and large projects and in its compatibility with many existing systems and protocols.

Git has been designed to support branching and tagging as first-class citizens (unlike SVN) and operations that affect branches and tags (such as merging or reverting) are also stored as part of the change history. Not all version control systems feature this level of tracking.

How does Git work? (1/2)

The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data.

Conceptually, most other systems store information as a list of file-based changes. These systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they keep as a set of files and the changes made to each file over time.

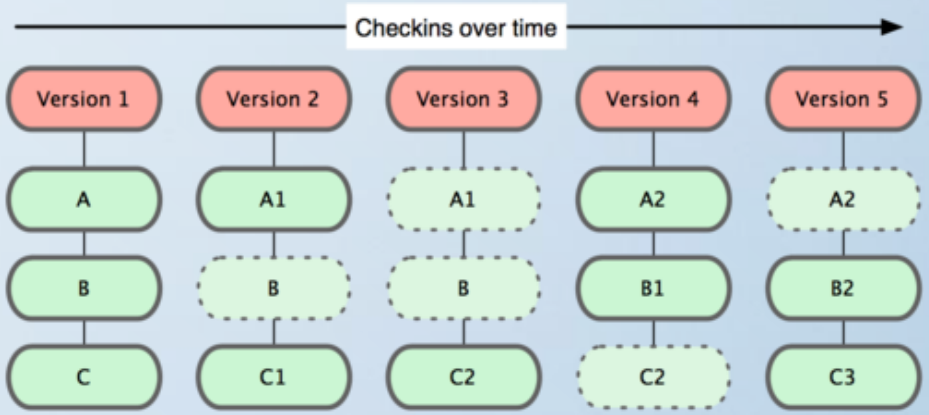


How does Git work? (2/2)

Git thinks of its data more like a set of snapshots of a mini filesystem.

Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again—just a link to the previous identical file it has already stored.

Git thinks about its data more like as below:



Git

Initial release: 7 April 2005

Written in: C, Shell, Perl, Tcl and Python

Operating system: Linux, Windows, OS X

Platform: POSIX

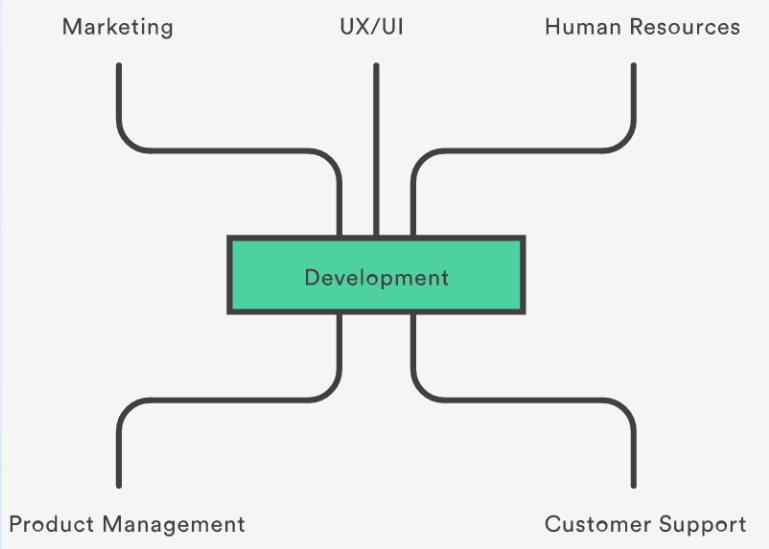
Type: Version control

License: GNU GPL v2 and GNU LGPL v2.1

Website: https://git-scm.com

Who needs Git?

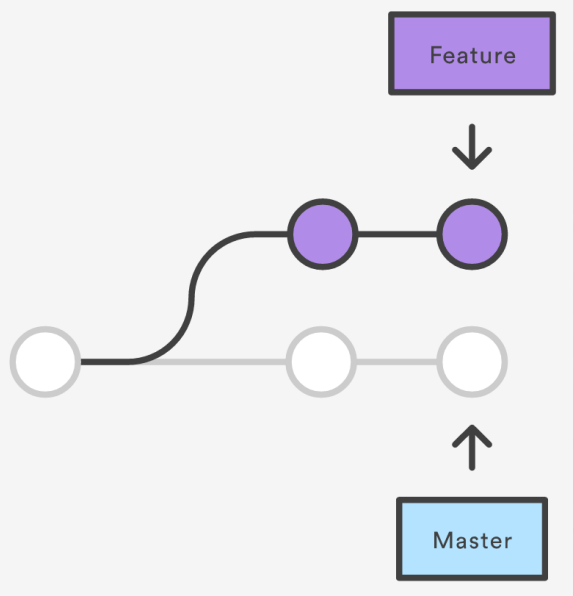
Switching from a centralized version control system to Git changes the way your development team creates software. And, if you're a company that relies on its software for mission-critical applications, altering your development workflow impacts your entire business.



Git for Developers (1/5)

Feature Branch Workflow

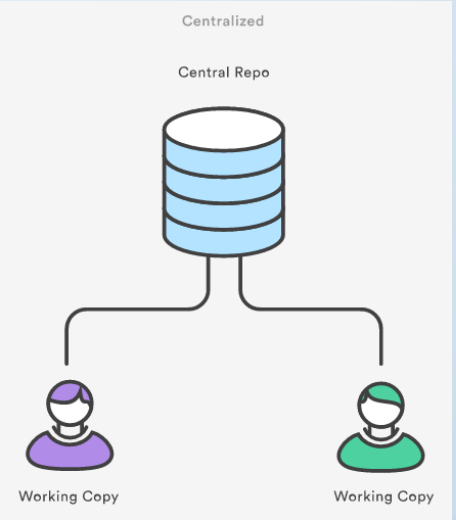
- One of the biggest advantages of Git is its branching capabilities. Unlike centralized version control systems, Git branches are cheap and easy to merge. This facilitates the feature branch workflow popular with many Git users.
- Feature branches provide an isolated environment for every change to your codebase. When a developer wants to start working on something—no matter how big or small—they create a new branch. This ensures that the master branch always contains production-quality code.

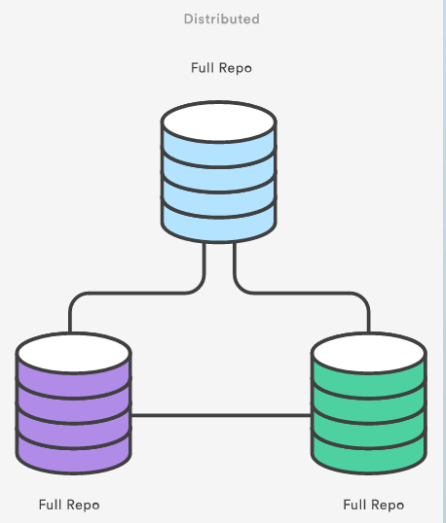


Git for Developers (2/5)

Distributed Development

- In SVN, each developer gets a working copy that points back to a single central repository. Git, however, is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits.
- Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.

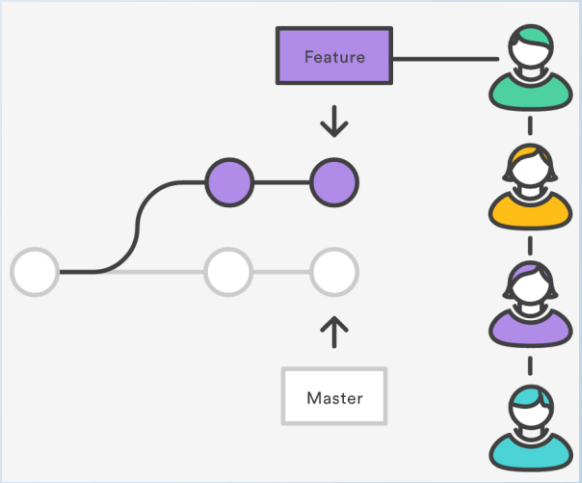




Git for Developers (3/5)

Pull Requests

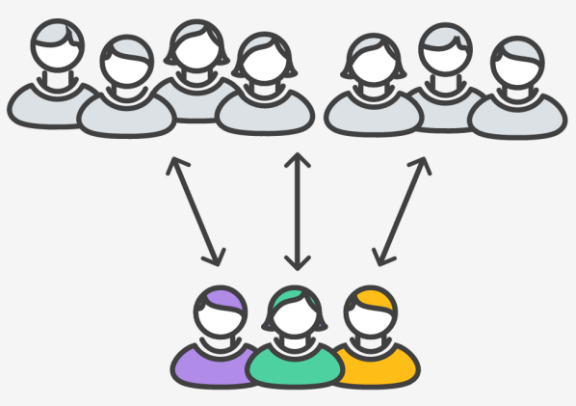
- A pull request is a way to ask another developer to merge one of your branches into their repository. This not only makes it easier for project leads to keep track of changes, but also lets developers initiate discussions around their work before integrating it with the rest of the codebase.



Git for Developers (4/5)

Community

- In many circles, Git has come to be the expected version control system for new projects. If your team is using Git, odds are you won't have to train new hires on your workflow, because they'll already be familiar with distributed development.
- In addition, Git is very popular among open source projects. This means it's easy to leverage 3rd-party libraries and encourage others to fork your own open source code.



Git for Developers (5/5)

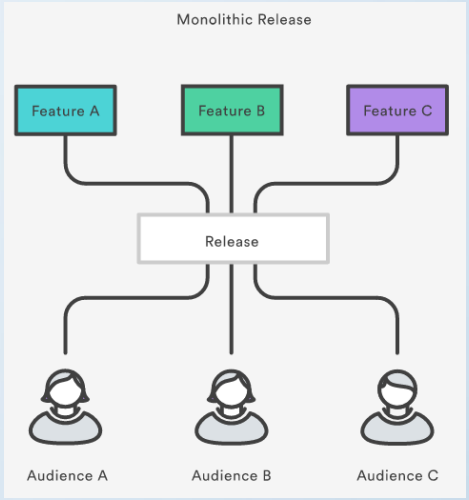
Faster Release Cycle

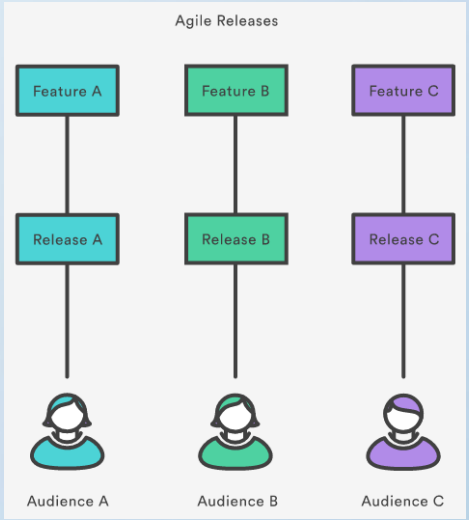
- The ultimate result of feature branches, distributed development, pull requests, and a stable community is a faster release cycle. These capabilities facilitate an agile workflow where developers are encouraged to share smaller changes more frequently. In turn, changes can get pushed down the deployment pipeline faster than the monolithic releases common with centralized version control systems.

Git for Marketing

If you're using a traditional development workflow that relies on a centralized VCS, all of these changes would probably be rolled up into a single release. Marketing can only make one announcement that focuses primarily on the game-changing feature, and the marketing potential of the other two updates is effectively ignored.

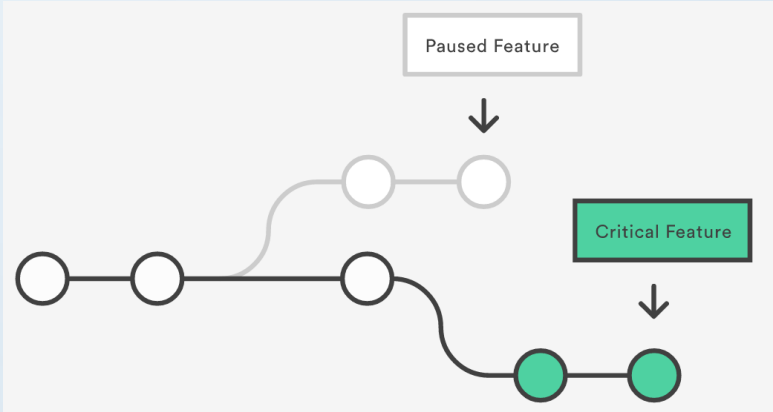
The shorter development cycle facilitated by Git makes it much easier to divide these into individual releases. This gives marketers more to talk about, more often. In the above scenario, marketing can build out three campaigns that revolve around each feature, and thus target very specific market segments.





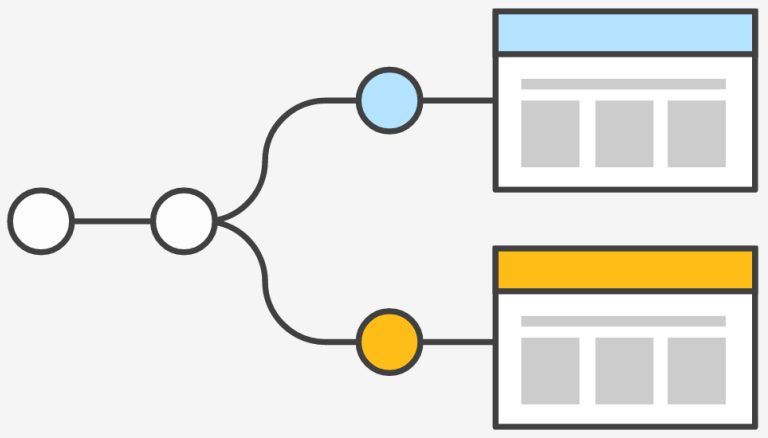
Git for Product Manager

The benefits of Git for product management is much the same as for marketing. More frequent releases means more frequent customer feedback and faster updates in reaction to that feedback. Instead of waiting for the next release 8 weeks from now, you can push a solution out to customers as quickly as your developers can write the code.



Git for Designers

Feature branches lend themselves to rapid prototyping. Whether your UX/UI designers want to implement an entirely new user flow or simply replace some icons, checking out a new branch gives them a sandboxed environment to play with. This lets designers see how their changes will look in a real working copy of the product without the threat of breaking existing functionality.



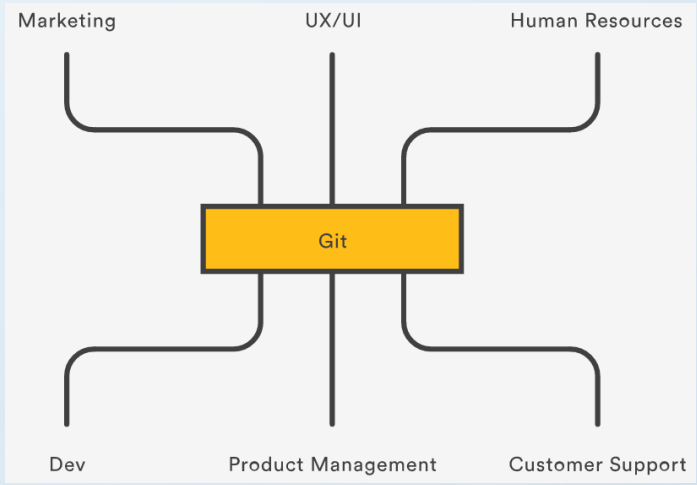
Git for Customer Support

Customer support and customer success often have a different take on updates than product managers. When a customer calls them up, they're usually experiencing some kind of problem. If that problem is caused by your company's software, a bug fix needs to be pushed out as soon as possible.

Git's streamlined development cycle avoids postponing bug fixes until the next monolithic release. A developer can patch the problem and push it directly to production. Faster fixes means happier customers and fewer repeat support tickets. Instead of being stuck with, "Sorry, we'll get right on that" your customer support team can start responding with "We've already fixed it!

Git for anyone managing a budget

Git is all about efficiency. For developers, it eliminates everything from the time wasted passing commits over a network connection to the man hours required to integrate changes in a centralized version control system. It even makes better use of junior developers by giving them a safe environment to work in. All of this affects the bottom line of your engineering department.



Installing Git on Windows

1. Download the latest Git for Windows installer from https://git-for-windows.github.io
2. When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation.
3. Open a Command Prompt (or Git Bash if during installation you elected not to use Git from the Windows Command Prompt).
4. Run the following commands to configure your Git username and email using the following commands:

```
$ git config --global user.name “FirstName LastName"
$ git config --global user.email “your-email@domain.com"
```


Installing Git on Mac

1. Download the latest Git for Mac installer from https://sourceforge.net/projects/git-osx-installer/files/
2. Follow the prompts to install Git.
3. Open a terminal and verify the installation was successful by typing git –version
4. Run the following commands to configure your Git username and email using the following commands:

```
$ git config --global user.name “FirstName LastName"
$ git config --global user.email “your-email@domain.com"
```


Installing Git on Linux

1. From your shell, install Git using apt-get:

   $ sudo apt-get update
   $ sudo apt-get install git

2. Verify the installation was successful by typing git --version:

3. Run the following commands to configure your Git username and email using the following commands:

   $ git config --global user.name “FirstName LastName"
   $ git config --global user.email “your-email@domain.com"

Git repository hosting services (1/2)

Assembla	Beanstalk	Bitbucket	CloudForge
Codebase	Fog Creek Kiln	GitHub	GitLab
Microsoft Visual Studio Team Services	planio	Perforce	RhodeCode

Git repository hosting services (2/2)

Features to consider

- Git
- Code Review
- Issue Tracker

Introduction to GitHub

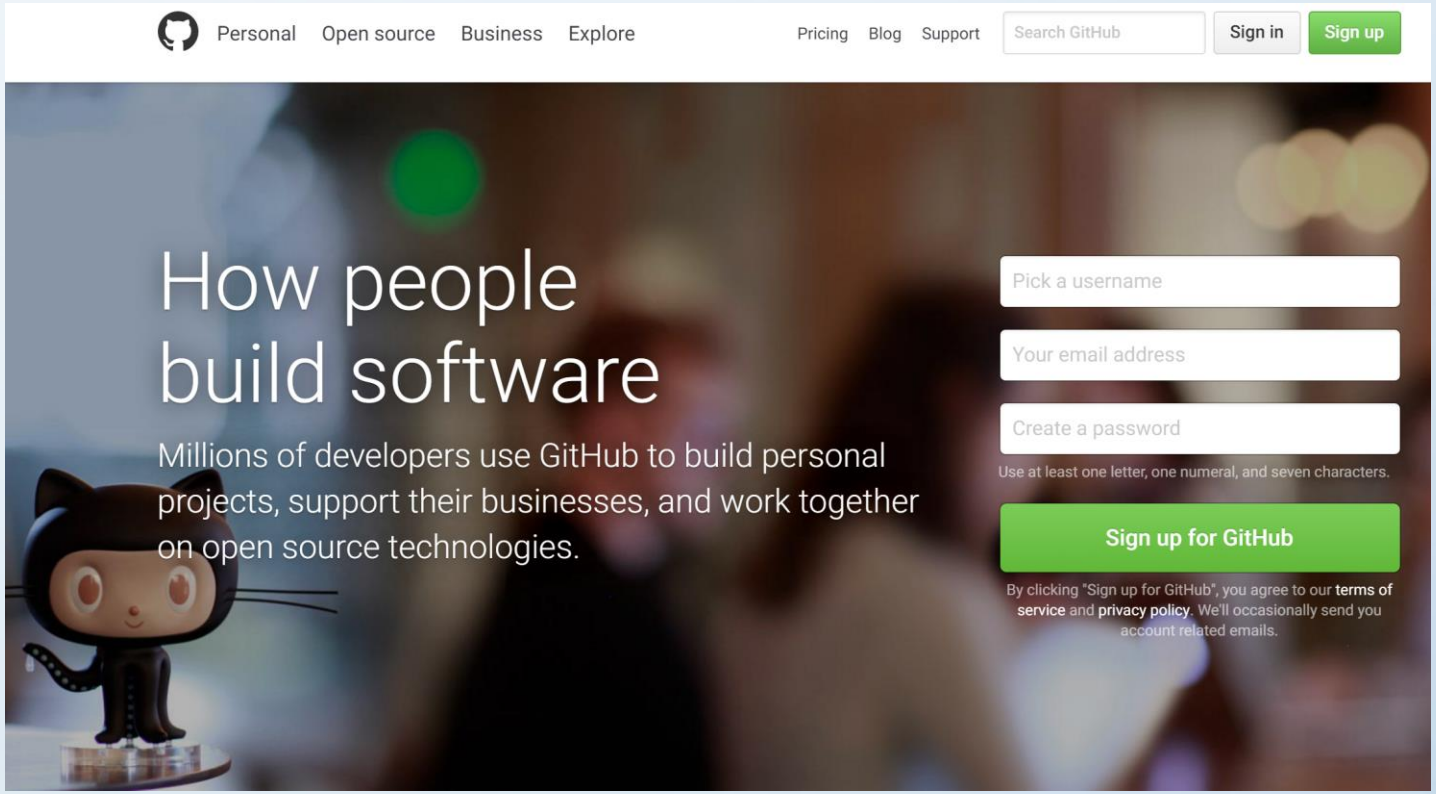
GitHub is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. GitHub offers both plans for private repositories and free accounts.

Website: https://github.com

GitHub

Introduction to GitHub

Create your account in GitHub.



Git GUI Clients

GitHub Desktop

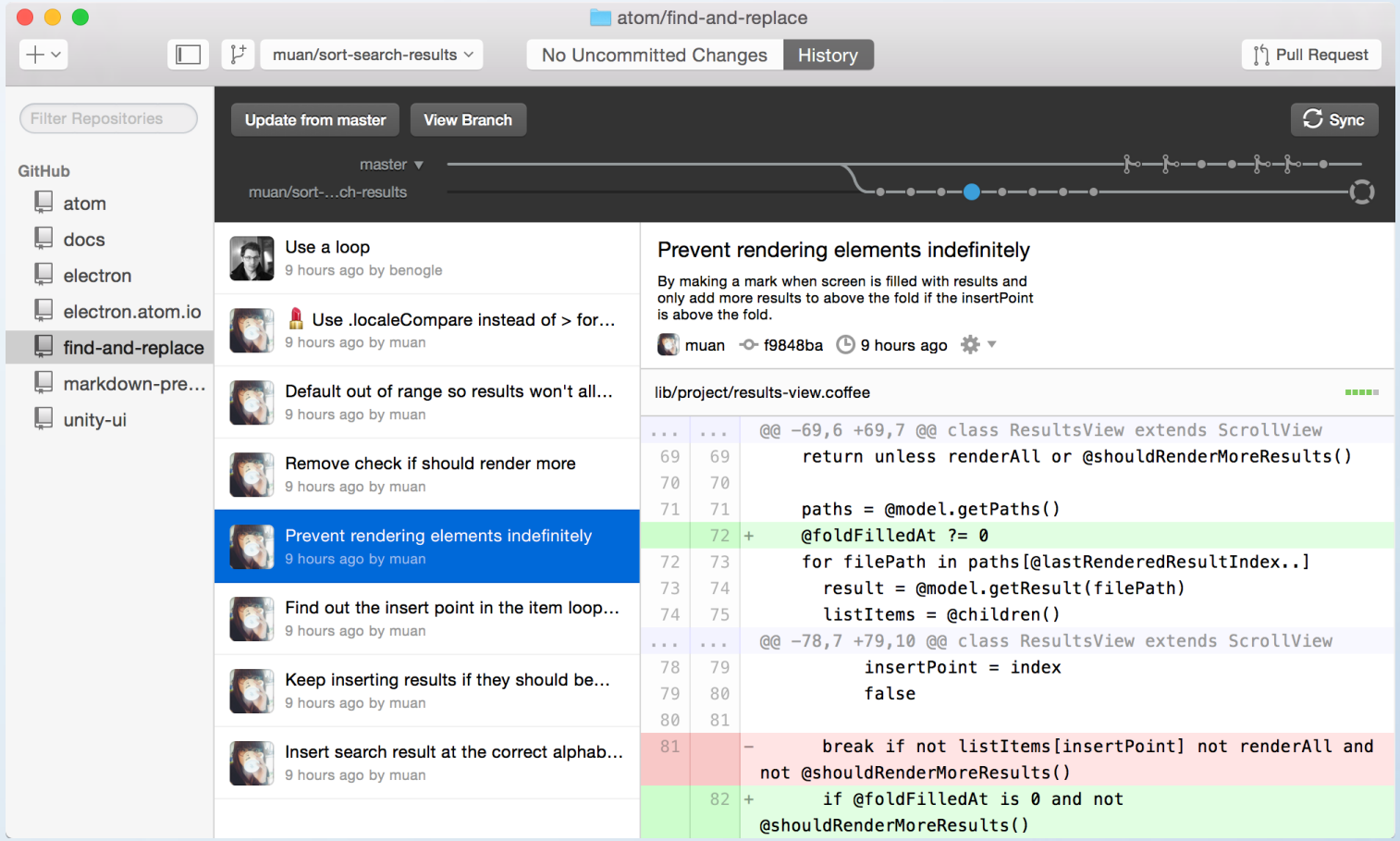
Tower

SourceTree

Git Extensions

Check the full list from here: https://git-scm.com/downloads/guis

Git GUI Clients – GitHub Desktop



Further study

1. A Visual Guide to Version Control (S10)
   - https://betterexplained.com/articles/a-visual-guide-to-version-control/
2. Introduction to Continuous Integration (S22)
   - https://www.youtube.com/watch?v=4sANX9AhM8c

Contribute

Feel free to contribute to make the content of the course even more beneficial and practical for all the students!

TechClass

Thank you for your consideration!

I hope you have a wonderful class! ☺

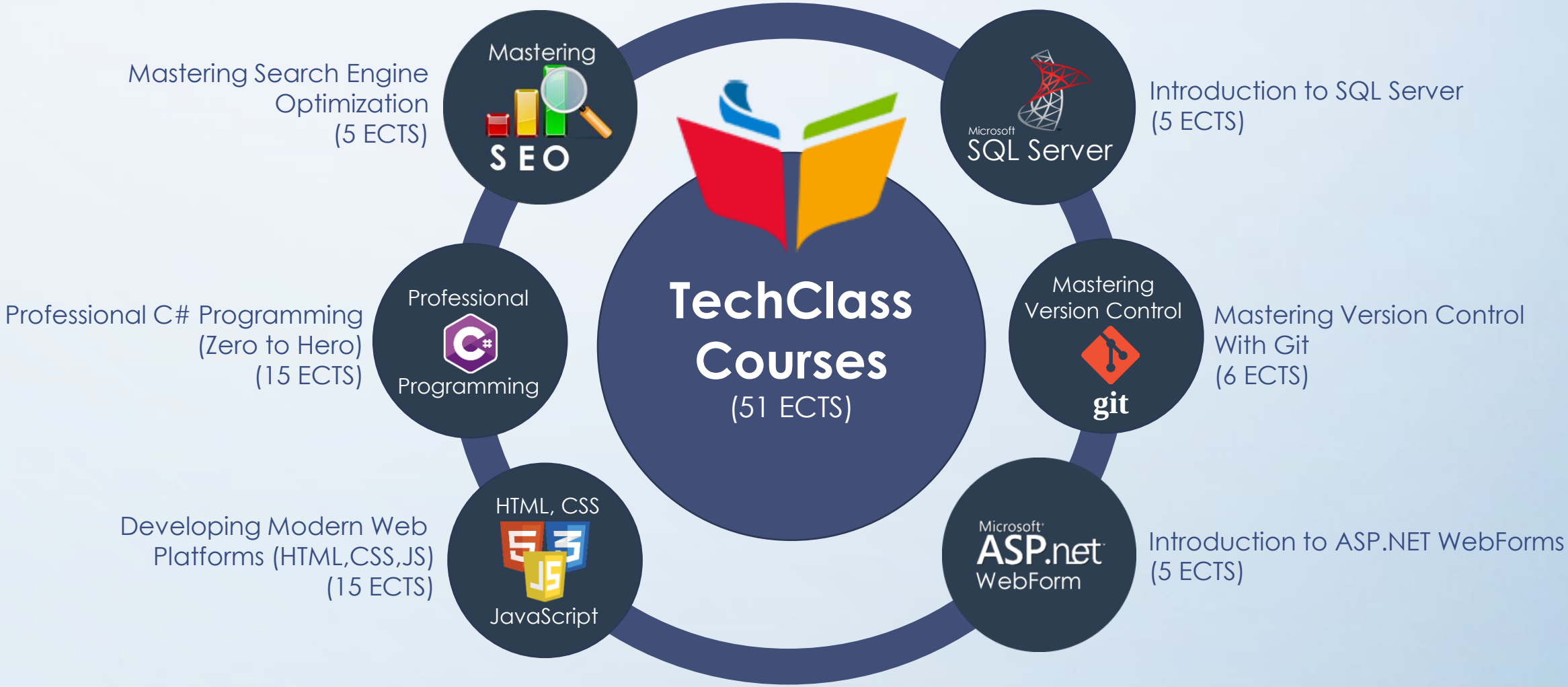
fb.com/techclass

@etechclass

@etechclass

bit.ly/etechclass

Helsinki Metropolia UAS
Bulevardi 31
00079 Heksinki, Finland
www.techclass.co


Mastering Search Engine Optimization (5 ECTS)
Mastering
SEO
Introduction to SQL Server (5 ECTS)
Microsoft
SQL Server
Professional C# Programming (Zero to Hero) (15 ECTS)
Professional
Programming
TechClass Courses
(51 ECTS)
Mastering Version Control With Git (6 ECTS)
Mastering
Version Control
git
Developing Modern Web Platforms (HTML,CSS,JS) (15 ECTS)
HTML, CSS
JavaScript
Introduction to ASP.NET WebForms (5 ECTS)
Microsoft
ASP.net
WebForm