Mastering Azure Repos

Accentient™

Azure DevOps Services | 2022.09

ACCENTIENT EDUCATION SERIES

Committed to training success

www.accentient.com

Mastering Azure Repos

Course Number:	**MARS**
Version:	**2022.09**
Software version:	**ADSvc**

Disclaimer

While Accentient takes great care to ensure the accuracy and quality of these materials, all material is provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Mastering Azure Repos

Course Introduction

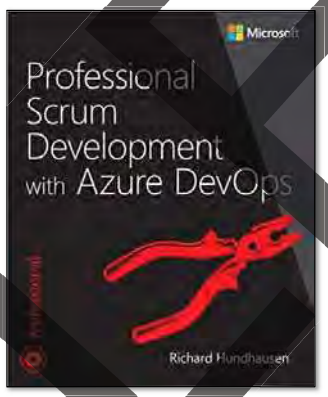
Accentient

- A leader in ALM, DevOps, and Scrum knowledge
- Helped thousands of teams and individuals understand and implement Azure DevOps/VSTS/TFS and Scrum successfully
- Has a close working relationship with Microsoft
- Course creator and steward for Scrum.org
- Has trainers that are Microsoft MVPs, Professional Scrum Developers, Professional Scrum Trainers, and authors

www.accentient.com | @accentient

Course Creator: Richard Hundhausen

- President of Accentient
- Author of software development books
- First Microsoft TFS/ALM/DevOps MVP
- Professional Scrum Developer
- Professional Scrum Trainer
- Co-creator of Nexus scaled Scrum Fx
- richard@accentient.com
- @rhundhausen



http://bit.ly/PSDAzDo

PSD PST

Accentient

Prerequisites

- Familiarity with:
  - Software development lifecycle
  - Team based development
  - Agile software development
  - Version control basics
- Have used:
  - A modern version of Visual Studio
  - Azure DevOps/VSTS/TFS

Accentient

Team Formation

5 MIN

- Form into teams of five (5) members or less
- Make sure experts aren't all on the same team …
  - Git experts
  - Visual Studio experts
  - Azure DevOps/VSTS/TFS experts
- Collocate your team
  - Physically or virtually
- Name your team

Accentient

Introductions

- Name
- Title/Role
- Development Experience
- Azure DevOps/VSTS/TFS Experience
- Expectations

Accentient

Course Overview

- This course shows you how to configure and use Azure Repos to improve team collaboration and code quality
  - Introduction to Azure Repos
  - Basic Git workflows
  - Visual Studio integration
  - Working with Azure Repos
  - Advanced Git workflows and concepts

Course Backlog

1. Introduction to Azure Repos
   - Overview of Azure DevOps, Azure Repos
   - Creating and configuring a project
   - Creating and configuring a repository
2. Git Concepts
   - Overview of DVCS and Git
   - Cloning a repository
   - Basic Git workflows
3. IDE Integration
   - Visual Studio Integration
   - Visual Studio Code Integration
   - Connecting to Azure Repos
   - Cloning and opening repos
   - Basic Git workflows revisited
4. Working with Azure Repos
   - Reviewing and editing history
   - Comparing changes
   - Moving, renaming, reverting
   - Tagging
   - Branching, merging
   - Pull requests, code reviews
   - Rebasing
5. Mastering Azure Repos
   - Resetting, reverting, and rewriting history
   - Forking
   - Branch policies
   - GitHub integration
   - Advanced workflows
   - Scalar tools and extension

Our Azure DevOps Services Environment

- We will be using a shared instance of Azure DevOps Services
- Each team will …
  - Be collocated (physically or virtually)
  - Have its own Azure DevOps project
  - Collaborate on all work in this class
- Each team member will …
  - Need a Microsoft Account

Accentient

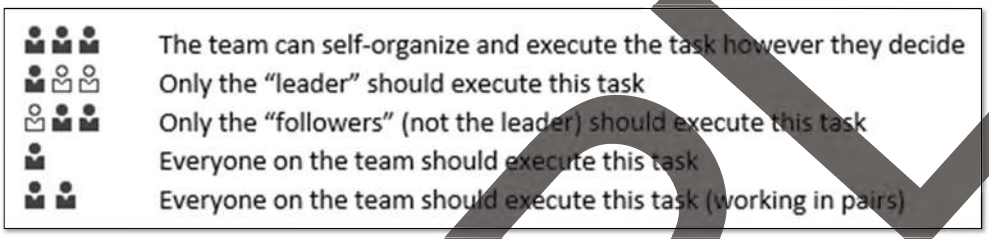
Schedule and Logistics

- Breaks
  - When should we have breaks?
- Labs
  - Labs can be breaks too
- Lunch
  - When should we break for lunch?

Accentient

Collaborating as a Team

- There are many opportunities for collaboration in this course
  - Some tasks, however, must be performed by *one* team member
- All tasks will be marked with an appropriate icon …



Accentient

Mastering Azure Repos

Module 1

Introduction to Azure Repos

Module Backlog

- Azure DevOps Services
  - Azure Repos
- Azure DevOps Projects
  - Creating
  - Configuring
- The Repository
  - Creating and configuring
  - Planning
- Lab

Azure DevOps Services

Azure DevOps Services

- An Azure-hosted SaaS alternative to on-premises Azure DevOps Server/TFS
  - Accessible from anywhere, using existing and familiar tools
  - Get started quickly – no infrastructure to manage
  - Backed by a 99.9% SLA
  - Monitored by a 24/7 operations team
  - Available in local data centers around the world
- Services
  - Boards, Pipelines, Repos, Test Plans, and Artifacts

Visit https://azure.com/devops for details

Azure Repos

- An Azure DevOps Service for hosting unlimited repositories
  - Git or TFVC version control systems
  - Private or public*
  - Social code reviews
- Pipeline integration
  - Build/Release (CI/CD)

* Not on Azure DevOps Server

Visit http://bit.ly/37e9mil for more information

Accentient

Azure Repos Supports Git

- Git is a distributed (decentralized) version control system
  - Each developer has a copy of the entire source repository on their dev machine
  - Developers can commit each set of changes on their dev machine and perform version control operations such as history and compare without a network connection
- Git is the most commonly used version control system
  - It has become the standard for version control
  - Git has support across Linux, Mac, and Windows platforms

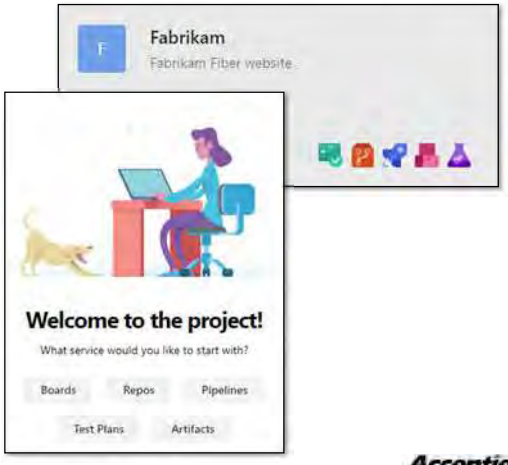
Visit http://git-scm.com for more information

Accentient

Azure DevOps Projects

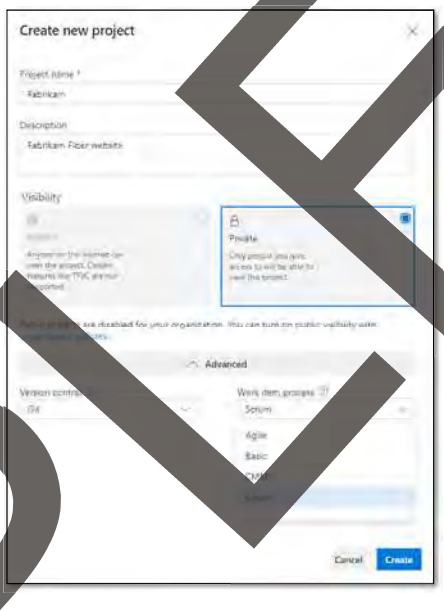
The Azure DevOps Project

- A project is a container where a development team can plan, track progress, and collaborate on building software
- It is a collection of:
  - Team members
  - Work items, backlogs, boards
  - Repositories and code
  - Build and release pipelines
  - Test plans and test cases
  - Artifacts
  - Workflows, policies, rules, metrics



Creating a Project

- Creating a new project …
  - Provide a short, meaningful name
  - Provide a description
  - Select public or private visibility (only applies to Azure DevOps Services)
  - Choose version control system
  - Choose work item process



Accentient

Configuring a Project

- Enabling/disabling services
- Teams, profiles, membership, permissions
- Notifications and dashboards
- Work item backlogs, iterations, areas, templates
- Repository permissions, behaviors, policies, options
- Build and release pipeline agent pools, behaviors, options
- Service hooks and service connections

Accentient

Public Projects

- Allow public or anonymous users limited access to the project's artifacts and services
  - You must enable anonymous access in Organization settings
- A private project's visibility can be changed to public
- Visit http://bit.ly/2DV0jrH for a complete list of the limited artifacts and services available in a public project

Accentient

The Repository

Accentient

The Repository

- A collection of folders, files, and its complete change history
  - Azure Repos hosts repositories
- Projects can have multiple repositories
  - You'll have a default one with the same name as the project
- Code can be added to an empty repository by …
  - Pushing a remote repository
  - Importing a repository (e.g. from GitHub)
  - Initializing manually with a README or gitignore file

Visit http://bit.ly/2RkmTh4 for more information

Mono-Repo or Multi-Repo?

- A project can have 1, 2, 10, or 100+ repositories
  - It's best to let the codebase decide
- Mono-Repo (a single, multi-package repository)
  - Teams with a monolithic codebase may want this
- Multi-Repo (multiple, single-package repositories)
  - The Skype team has hundreds of small repositories that get stitched together in various combinations to create their many different clients, services, and tools
  - Teams embracing microservices may want this
  - Tip: Avoid cyclical dependencies between repos

Mono-Repo

Pros	Cons
✓ Related changes made in one location	× May contain unrelated products/projects
✓ Simpler access control	× Fine-grained access control more complex
✓ Simpler organization and cloning	× Grows large faster/difficult to clone
✓ Simpler code and history browsing/searching	× Fine-grained versioning more complex
✓ Enables broader refactoring	
✓ Simpler dependency management	
✓ Simpler package versioning	
✓ Trivial to split into multiple repos (in the future)	

Accentient

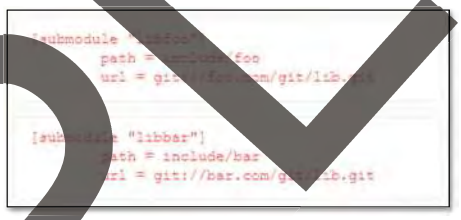
Multi-Repo

Pros	Cons
✓ Fine-grained access control	× More complex access control
✓ Smaller repos to clone	× More repos to manage
✓ More flexibility for tools and frameworks	× Building/testing entire product is difficult
✓ Clear ownership/stewardship	× Easier to introduce non-evident bugs
✓ Simpler pipeline management (per product)	× Complex code and history browsing/searching
	× Cross-repo refactoring is more difficult
	× Cross-repo versioning is more difficult
	× Difficult to merge multiple repos (in the future)

Accentient

Git Submodules

- Git submodules allow you to include external repositories in a repository using a *linking* approach
  - You can reference a specific commit in that external repository
  - This is useful when you have multiple repositories
- Submodules are defined in the .gitmodules file …
- Submodules are a good fit for component-based development, where your main project depends on a fixed version of component



Accentient

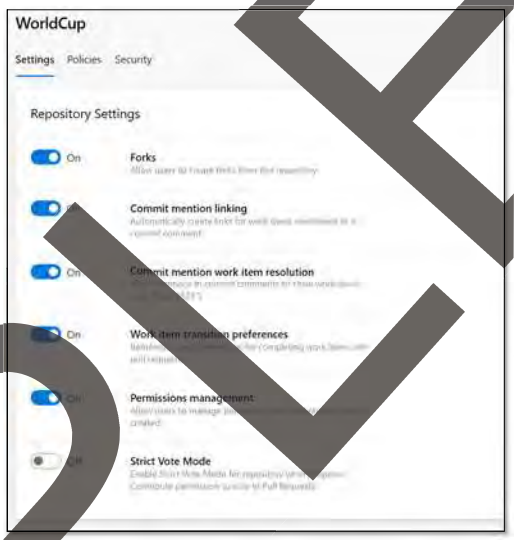
Git Subtree

- An Alternative to Git Submodules that lets you nest one repo inside another as a sub-directory using a copy approach
  - Easier management with a simpler workflow
  - All code is available right after the clone
  - Does not add new metadata files (e.g. .gitmodule)
  - Users can be ignorant of the fact that git subtree is being used
  - Contents of the module can be modified without having a separate repository copy of the dependency somewhere else

Accentient

Configuring Azure Repos

- Settings
  - Forks
  - Work item integration
- Policies
  - Default behaviors
  - Branch policies
- Security
  - Permissions applied to all repositories (via inheritance) or to specific repositories
  - Permissions to specific branches or tags within a repository



Accentient

Configuring Repository Permissions

- Permissions follow an inheritance model
  - Individual repositories inherit permissions from the top-level *Git Repositories*
  - Branches inherit permissions from assignments made at the repository level
- Tip: Set permissions at the top level *Git Repositories* entry and then tweak permissions for any specific repository, branch, or tag as needed

Accentient

Migrating TFVC to Git

- You can migrate code from an existing TFVC repository to a new Git repository within the same organization
  - Requires TFS 2017 Update 2 or later version
  - Import up to 180 days of history and not more than 1GB
  - Recommendation: Don't migrate TFVC history
  - Visit https://bit.ly/TFVCtoGit for more information
- For larger, more complex, migrations consider git-tfs
  - Visit https://github.com/git-tfs/git-tfs for more information

Accentient

Module Retrospective

What have we learned in this module?

- Azure DevOps offers many services that are helpful to a team that is planning, developing, and delivering software
  - Azure Repos hosts private or public repos for TFVC or Git
- Azure DevOps projects are the container for a product's application development lifecycle
  - Projects can be private or public (to enable public repos)
- Repositories can be secured and configured in many ways

Accentient

Lab

In this lab you will create and configure an Azure DevOps project and a Git repository.

- Setup the learning environment
- Create an Azure DevOps project
- Create a repository
- Create a public project (optional)

Accentient

Mastering Azure Repos

Module 2

Git

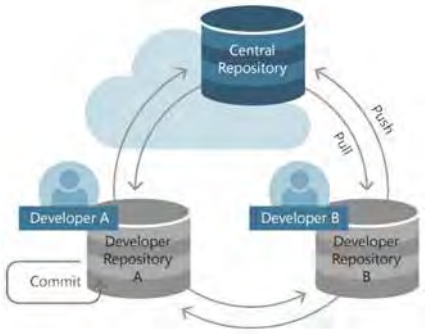
Module Backlog

- Git overview
  - Git for Windows
- Gitting started
  - Cloning a repository
  - Basic Git workflow
- Lab

Git Overview

Distributed Version Control System (DVCS)

- Your local copy of code is a complete version control repository
  - This differs from a *centralized* VC system where clients must synchronize code with a server before creating new versions of code
- Attributes of a DVCS …
  - Easy to work offline or remotely
  - No need to connect to a common server to access the full version history
  - Every developer's codebase contains a full history of ALL version changes
  - Every developer's local copy acts as its own fully-functional repository that does not need any other copies
  - Fast! (with the exception of pushing/pulling)



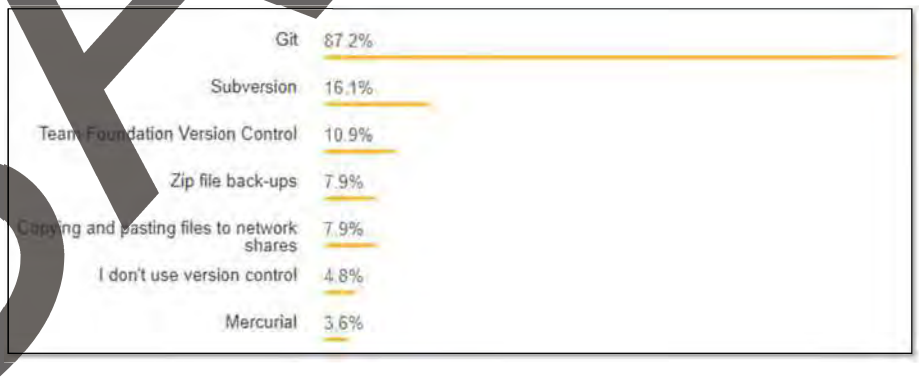
What is Git

- Git was created in 2005 by Linus Torvalds
  - Trivia: Linus named "Git" after the British slang word for a horrible person, saying that he named it after himself!
- Linus created Git for the development of the Linux kernel
  - Other kernel developers contributing to its initial development
  - Git's current maintainer since 2005 is Junio Hamano
- Free and open-source (GNU GPL v2)

Accentient

Git is Popular

- Git has become the standard version control system
- Almost 90% of developers use Git
  - According to StackOverflow's 2018 survey …



Accentient

Benefits of Git

- Simultaneous development
- Faster releases
- Built-in integration
- Strong community support
- Git works with your team
- Pull requests
- Branch policies
- Security

Visit https://bit.ly/2RWe2ym for more information

Accentient

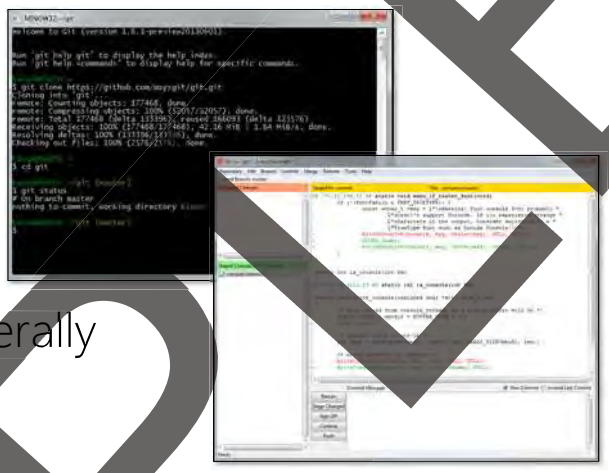
Git is Secure

- Git checks the boxes with *integrity* and *availability* in the CIA triad of security
- Integrity
  - Ensuring that information has not been altered
  - Git uses the SHA1 hashing algorithm to save version history
  - This hash is the basis for the names used in its history log
- Availability
  - Provided by the nature of a DVCS, ensuring that every developer gets a copy of the full version history

Accentient

Git For Windows

- Git for Windows is an open source project providing a lightweight, native set of tools supporting all Git features through various interfaces
  - Git BASH
  - Git GUI
  - Windows Shell Integration
- Releasing of new versions generally follow Git's release cycle



Visit https://gitforwindows.org for more information

Accentient

Common Git Commands

git add *	git fetch	git reflog
git branch	git init	git remote
git checkout	git log	git revert
git clean	git merge	git stash
git clone	git pull *	git status
git commit *	git push *	git tags
git config	git rebase	

* Development team fundamental Git workflow

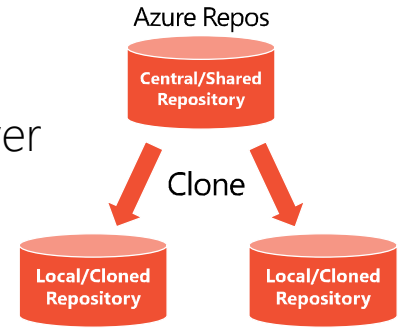
Accentient

Gitting Started

Accentient

Clone

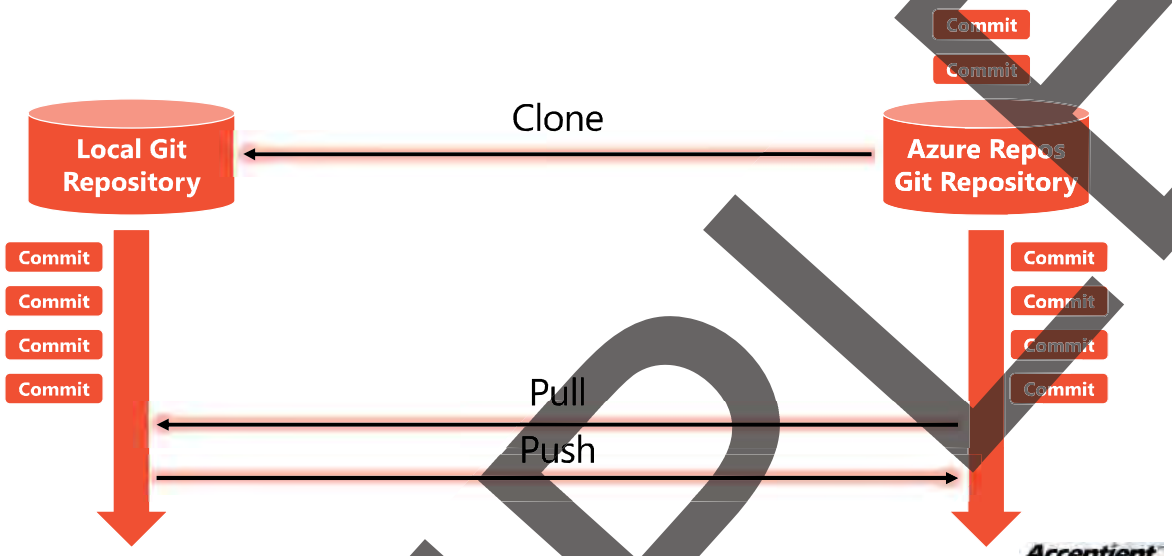
- Creating a local copy of an existing repository
  - The resulting repository can be referred to as a *clone*
- Typically there is a central (remote) server for keeping a central repository
  - Each cloned repository is a full copy of this repository, including all commits and branches



Visit http://bit.ly/2ow0fGL for more information

Accentient

Basic Workflow



Git Add

- Git *add* indicates that one or more changes in the working directory should be *staged*
  - In other words, it tells Git that you want to include the update(s) to a particular file in the next commit
- Git add does not actually update the repository
  - Changes are not actually recorded until *commit* is executed
- Use the *git reset* command to *unstage* any staged changes

Visit http://bit.ly/2UbLDda for more information

Git Status

- Git *status* displays the state of the working directory and the staging area
- You can see …
  - Which changes have been **staged**
  - Which changes are **unstaged**
  - Which files are **untracked**
- Status output does not show you any information regarding the committed project history
  - Use git *log* for that information

Visit http://bit.ly/2zECSjm for more information

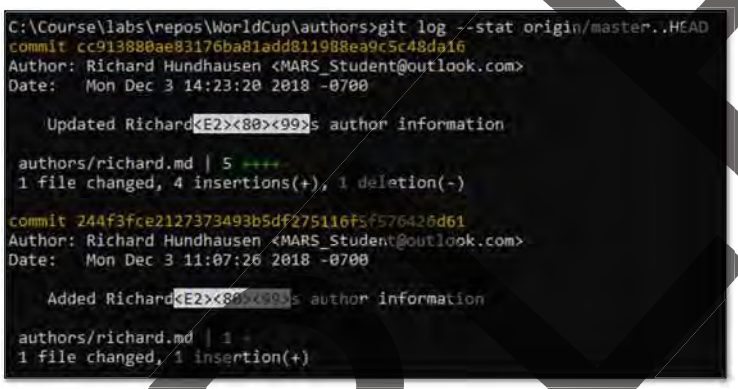
Git Commit

- Git *commit* is used to commit (save) a snapshot of the staging directory to the repository
  - In Git, a commit is the equivalent of a "save"
- Commits can be accrued locally, then pushed to a remote repository (Azure Repos) as needed using the git *push* command

Visit http://bit.ly/2AQQOXo for more information

Git Log

- Git *log* lists the committed snapshots in the project history
  - It can be displayed in various formats
  - searched or filtered



Visit http://bit.ly/2AQHK4r for more information

Accentient

Undoing Commits and Changes

- There are many "undo" related commands and strategies"
- Git **checkout**
  - Restore older working tree files
- Git **revert**
  - Revert existing commits
- Git **reset**
  - Reset current HEAD to the specified state
- Git **clean**
  - Remove untracked files from the working tree
- Git **rm**
  - Remove files from the working tree and from the index

Accentient

Git Pull

- Used to download content from a remote repository into the local repository
  - This is a common task, part of Git-based collaboration work flows
- It is important to *pull* prior to *push*-ing changes to a remote repository to avoid conflicts
- Note: Git *pull* is actually a combination of *fetch* and *merge*
  - Git *fetch* downloads objects and refs from another repository
  - Git *merge* joins two or more development histories together

Visit http://bit.ly/2QgqTD3 for more information

Accentient

Git Push

- When a developer is ready to contribute locally-committed changes to the team's remote repository
  - If the team has pushed commits to the branch since the developer last cloned/pushed, all changes must be *pulled* prior to pushing any changes
  - In some cases, a developer may want to push to an alternate remote repository
- *Sync* is the combination of *pull* (*fetch* + *merge*) + *push*
  - *Sync* is not a Git command

Visit http://bit.ly/2QewKZs for more information

Accentient

Module Retrospective

What have we learned in this module?

- Git is a Distributed Version Control System (DVCS)
  - It is also the most popular version control system in use today
- Git repositories contain every version of every file saved therein
  - Git is efficient, so having a many files/versions doesn't waste disk space
- Put dependent codebases into the *same* repository
  - Consider putting *all codebases* into one repository (the mono-repo)
- Basic Git workflows
  - Local: *add* ⇨ *commit*, remote: *pull* ⇨ *push*

Accentient

Lab

30 Minutes

In this lab you will install and use Git for Windows to interact with a local repository as well as Azure Repos.

- Install Git for Windows
- Clone repository
- Add files and commit changes
- Undo changes
- Push to remote repository
- Review in Azure Repos

Accentient

Lab 1: Introduction to Azure Repos

Mastering Azure Repos

LAB OVERVIEW

This lab walks you and your colleagues through the process of forming into teams and provisioning Azure DevOps Services required for the rest of the course.

Estimated time to complete this lab: **30 minutes**

Task Execution

As this is a team-based training course, there are a number of opportunities for team members to learn to collaborate more effectively. Unfortunately, there is a possibility for team members to accidentally impede, block, or otherwise cause unintentional conflicts. To minimize the possibility of conflicts, critical tasks in this course have been marked with an icon indicating who on the team should execute the task:

-  The team can self-organize and execute the task however they decide
-  Only the "leader" should execute this task
-  Only the "followers" (not the leader) should execute this task
-  Everyone on the team should execute this task
-  Everyone on the team should execute this task (working in pairs)

Tip: Look for the "leader"  tasks and ensure that they are only performed once per team. Also, ensure that the "follower"  tasks are only performed by everyone else (not the leader).

Teams of One

If you are working by yourself and not on a team, make sure to perform all of the "leader"  tasks, and none of the "follower"  tasks. This scenario is common for students learning remotely.

EXERCISE 1 – SET UP THE ENVIRONMENT

Task: Install Courseware Files

In this task you will install the files required by this class.

Dependencies

- Signed in as *Administrator*

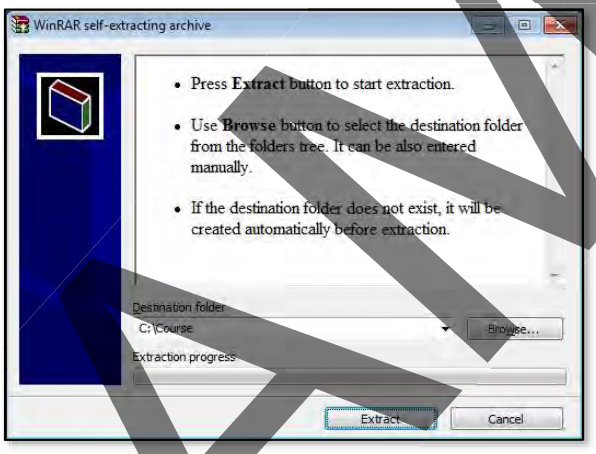
1. Verify that the **C:\Course** folder does not exist.

   Note: If this folder already exists, then your computer may have been used for a prior training class. If that is the case, then the effectiveness of the hands-on labs that follow may be diminished.

2. If necessary, copy the courseware file to your desktop.

   This file may already be on the desktop. If not, you may have to ask your instructor for help locating and/or copying this file. If you cannot locate this file, please email support@accentient.com to obtain a copy.

3. Extract the courseware files, specifying **`C:\Course`** as the **Destination folder**.



   It can take a few moments to extract the files. The folder C:\Course will be created during the process. After extracting the files, you should have one or more of the following sub-folders:

- C:\Course\Guidance
- C:\Course\Labs
- C:\Course\Software

EXERCISE 2 – SET UP YOUR TEAM

Task: Form Into Teams (optional)

If necessary, your instructor will facilitate the creation of equally sized (5 team members or less), cross-functional, collocated teams.

- Form into cross-functional teams (of 5 members or less)
- Do your best to ensure the experienced "experts" aren't all on the same team …
  - Git experts
  - Azure DevOps experts
  - Visual Studio experts
  - Visual Studio Code experts
- Collocate (to the best of your abilities)
- Introduce yourself (if necessary)
- Decided on a team name (i.e. "Team Blue", "The Honey Badgers", "Repo Team", etc.)
- Write your name and team name where it is visible to others in the class

What is the name of your team? _____

Who are your team members? _____

_____

Task: Identify a Microsoft Account

In this task you will identify, or create if necessary, a Microsoft Account that you will use for the various online services leveraged during this class.

What is your Microsoft Account (MSA)? _____

1. If you already have a Microsoft Account, write it on the line above and skip the rest of these steps.
2. Open **Chrome** and navigate to https://signup.live.com.

   You may want to use Incognito mode. Also, if you don't have Google Chrome installed, you can install it from here: www.google.com/chrome.
3. Provide the required information.

   This will include your name, username, password, country/region, zip code, birthdate, and gender. You can create a new *outlook.com* or *hotmail.com* address. You will also need to provide a phone number or other method to be able to reset your password.
4. **Create** the account, accepting the Microsoft Services Agreement and privacy statement.

Task: Create an Azure DevOps Services Organization and Project

In this task your team will select someone to create a new Azure DevOps Services Organization and project.

Who will be performing this task? _____

1. Launch **Chrome**, navigate to https://dev.azure.com, and get started for free.

   You may be asked to review and accept the licensing terms.

   What is the name of your new organization? _____

2. With the new organization selected, select **Organization settings**.

   On the *Overview* page, what is the *URL*? _____

   If you are happy with this organization name and URL, then skip the next step.

3. Rename the organization (and thus the URL) to a better, more meaningful name.

   Tip: Consider using a variation of your team's name.

   What is the new Organization? URL? _____

4. Share the **Organization URL** with your team members.

5. Change the **Time zone** accordingly.

6. Go to the **Projects** page and create a **new project**:

   - Project name: **WorldCup**
   - Description: **2026 World Cup**
   - Visibility: **Private**
   - Version Control: **Git**
   - Work item Process: **Scrum**

7. On the **WorldCup** project page, select **Project settings**.

8. On the **Overview** page, turn off (remove) the **Test Plans** and **Artifacts** services.

9. Press **F5** to refresh the page.

   This will remove the corresponding service icons on the left side of the screen.

Task: Configure the Team

In this task the person who created the project will configure the default team.

1. In Project settings, go to the **Permissions** page.
2. Select the **WorldCup Team** group and click **Member of**.
3. **Add** the **[WorldCup]\Project Administrators** group.

   Note: In practice, you should check with your Azure DevOps administrator before adding all members of a team to the *Project Administrators* group.
4. Delete membership of the **Contributors** group.

   This membership is not necessary if team members are members of the Project Administrators.
5. Click **Members**.

   How many members are currently listed? _____
6. **Add** the **Microsoft Accounts** of your teammates.

   How many team members are listed? _____

   You may need to refresh the page. Also, Microsoft may send invitational emails to your colleagues. These emails can be ignored.

Task: Join the WorldCup Team

In this task the rest of the team members will access the newly created project.

Note: You may receive an invitational email from Microsoft. You can ignore this.

1. Open your browser and navigate to the URL of your project.

   Example: https://dev.azure.com/mars2022/worldcup
2. If prompted, enter your **Microsoft Account** and **password**.

   If you have any difficulties signing in, double-check your account and password. If that doesn't solve the problem, ask your colleague to double-check your team membership.

Task: Update Your Profile (optional)

In this task everyone will update their individual profile.

1. In the upper-right corner, click User settings and then select **Profile**.
2. If you want, change your **Profile Picture** and upload a more representative picture.

   Having team member pictures will improve your team's social experience.
3. Change to the correct **Time Zone**.
4. Opt out of all **Notifications**.

   There may be a slight delay as you click each toggle, but in the end, your inbox will thank you.

EXERCISE 3 – CREATE YOUR REPOSITORY

Task: Create a Repository

In this task your team will select someone to create a new Git repository in Azure Repos.

Who will be performing this task? _____

1. If necessary, navigate to the **WorldCup** project.
2. Go the **Files** page on the **Repos** hub.

   Your project should have no repositories and no code yet.
3. **Import** the **Git** repository found here: https://github.com/accentient/WorldCup2022.

Task: Review the Repository

In this task each team member, or pair of team members, will review the contents of the repository.

Dependencies

- The *WorldCup2022* GitHub repository has been imported to the WorldCup project

1. If necessary, navigate to the **WorldCup** project.
2. Go to the **Files** page on the **Repos** hub.
3. Locate and click on the top-level **README.md** file.

   This markdown file explains the purpose of the repository.
4. Review the folders and files under **src**.
5. Select the top level **WorldCup** folder (the repo root) in the tree and view **History**.

   How many *entries* are there? _____

   What was the *date* of the first commit? *Who* was the author? _____
6. Return to the **Files** page on the **Repos** hub.
7. Right-click on the src folder and select **Download as zip**.

   Individual files can be downloaded directly from Azure Repos. Folders are downloaded as zip files. Files can also be uploaded directly to the repository.

Task: Create Folder and Upload Files

In this task your team will self-organize and decide who will create a new folder and who will upload some graphic image files to the new repository.

Who will be creating the new folder (steps 2-4 below)? _____

Who will be uploading the files (steps 5-8 below)? _____

1. If necessary, navigate to the **Files** page on the **Repos** hub.
2. Select the ⋮ context menu to the right of the top level **WorldCup** folder and add a new **Folder** named **images** with a new file named **README.md** and click **Create**.

   Git folders cannot be empty, so a placeholder file will be added.
3. In the **README.md** contents, enter the following markdown:

   ```
   ###World Cup 2026 icons
   ```
4. **Commit** the changes with the default comment.
5. Have everyone refresh their browser.
6. Select the ⋮ context menu to the right of the new **images** folder and select **Upload files(s)**.
7. Browse to **C:\Course\Labs\Lab01\images** and select the images.
8. Enter a Comment of **Added candidate World Cup 2026 images** and **Commit** the changes.
9. Have everyone refresh their browser and review the new folder and files.

   You can click on the image files and view the images directly in the browser.

Task: Review Repository Settings

In this task each team member, or pair of team members, will review various repository settings.

1. Click on **Project settings**.
2. Select **Repositories**.

   This page allows you to view and manage the security settings and options for the various repositories.
3. Select the **WorldCup** repository.

   Can your team members create *Forks* from this repository? _____

   Is *Commit mention linking* enabled? _____

4. Enable all **Commit mention** … options.

5. View **Policies**.

   Are any *Repository Policies* enabled by default? _____

6. View **Security**.

   Is security *Inheritance* on or off? _____

   What are members of the *Readers* group allowed to do? _____

7. Select the **Contributors** group and review the permissions.

   Developers will typically be members of the *Contributors* group. In this class, however, you are all members of the *Project Administrators* group.

   Notice that security permissions can be assigned to pipelines, tags, and branches as well.

8. Return to the **Repositories** main page.

   This is the one that shows *All Repositories*. Settings, Policies, and Security changes will be inherited by each repository.

9. View **Settings**.

   What is the default branch name for new repositories? _____

10. View **Policies**.

    These are the top-level policies that will be inherited by each repository. Individual policies can be overridden.

EXERCISE 4 – CREATE A PUBLIC PROJECT (OPTIONAL)

Task: Create a Public Project

In this task the person who originally created your Azure DevOps Services organization will **create a new** public Azure DevOps project.

Who will be performing this task? _____

1. Navigate to your Azure DevOps Services organization homepage.

   Example: https://dev.azure.com/avengers.

2. Select **Organization settings**.

3. Go to the **Policies** page.

   Does your security policy currently A*llow public projects*?_____

4. Ensure that the **Allow public projects** policy is **On**.

5. Go to the **Projects** page and create a **new project**:

   - Project name: **Mascot**
   - Description: **World Cup mascot community development**
   - Visibility: **Public**
   - Version Control: **Git**
   - Work item Process: **Basic**

6. On the new **Mascot** project page, go to **Project settings**.

7. On the **Overview** page and turn off all services except for **Repos**.

   Notice also that you can toggle the visibility between public and private. Before you convert a private project to public, visit http://bit.ly/2KJUsXl to review a list of considerations.

8. Refresh the page.

   This will remove the corresponding service hub icons on the left side of the screen.

9. Go the **Files** page on the **Repos** hub.

   Your project has no repositories and no code yet.

10. **Import** a Git repository from https://github.com/accentient/WorldCupMascot.

11. Return to the **Overview** page and share the Mascot project URL with your team members.

    What is the URL? _____

Task: Review the Public Project or

In this task each team member, or pair of team members, will review the newly created public project.

Dependencies

- The *Mascot* public project has been created and you know the URL

1. Open a new Incognito Chrome session, or use a different browser than you are currently using.
2. Navigate to the URL of the new **Mascot** public project.

   Did Azure DevOps Services require you to *sign-in*? _____

   Do you see a *Public* badge in the upper right corner? _____

3. Go to the **Files** page on the **Repos** hub.
4. Browse to the **Previous World Cup mascots** folder and view the **mascots.jpg** file.

   As you can see, public projects allow anonymous users access to your repositories and other services. By default, this is read-only access, but that can be changed. To see a list of the default roles and permissions for public projects, visit http://bit.ly/2DSxJXK.

Lab 2: Git for Windows

Mastering Azure Repos

LAB OVERVIEW

This lab has you and your colleagues installing and using Git for Windows to experience basic Git workflows.

Estimated time to complete this lab: **30 minutes**

Task Execution

As this is a team-based training course, there are a number of opportunities for team members to learn to collaborate more effectively. Unfortunately, there is a possibility for team members to accidentally impede, block, or otherwise cause unintentional conflicts. To minimize the possibility of conflicts, critical tasks in this course have been marked with an icon indicating who on the team should execute the task:

-  The team can self-organize and execute the task however they decide
-  Only the "leader" should execute this task
-  Only the "followers" (not the leader) should execute this task
-  Everyone on the team should execute this task
-  Everyone on the team should execute this task (working in pairs)

Tip: Look for the "leader"  tasks and ensure that they are only performed once per team. Also, ensure that the "follower"  tasks are only performed by everyone else (not the leader).

Teams of One

If you are working by yourself and not on a team, make sure to perform all of the "leader"  tasks, and none of the "follower"  tasks. This scenario is common for students learning remotely.

EXERCISE 1 – INSTALL GIT FOR WINDOWS

Task: Install and Configure Git for Windows or

In this task each team member, or pair of team members, will download, install, and configure the latest version of Git for Windows.

1. Download the latest version of **Git for Windows** from https://git-scm.com/download/win.

2. Install **Git for Windows** leaving all of the default settings.

   This is a required component for this course. Visit http://bit.ly/2DTm7Uq for more information. As part of the default installation, the *Git Credential Manager for Windows* will be installed. Part of installing Git for Windows may be the *removal* of a previous version.

3. Open a command window and run the following command:

   ```
   git version
   ```

   What *version* is listed? _____

   If you have problems running the command, ensure that *C:\Program Files\Git\cmd* is in your PATH.

   Tip: Use the *Developer Command Prompt for VS 2022*

4. Run the following commands to see how to get help:

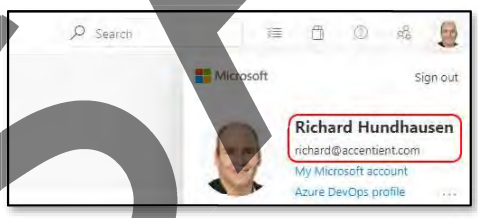
   ```
   git help
   git help config
   ```

   Most help is provided in the form of HTML documentation.

5. Return to the command window and run the following config commands:

   ```
   git config --global user.name "<user name>"
   git config --global user.email "<email address>"
   git config --global core.editor notepad
   ```

   The user name and email address will be the ones you used in Azure DevOps Services:

   We are also overriding the default editor (VIM) to use Notepad instead. You're welcome.

6. Run the following command to review the new global configuration settings:

   ```
   git config --global --list
   ```

Task: Clone the WorldCup Repository or

In this task each team member, or pair of team members, will clone their team's WorldCup repository.

1. Return to the browser and go to the **Files** page on the **Repos** hub of the **WorldCup** project.

   Important: Make sure you are in the *WorldCup* project and not the *Mascot* project.

2. Click **Clone** in the upper right and copy the **URL** to the clipboard.

3. Return to the command prompt and run the following commands:

   `git clone <clone URL> c:\course\labs\repos\WorldCup`

   Where *<clone URL>* is the url you copied to the clipboard above. Here is an example:

   git clone https://avengers@dev.azure.com/avengers/WorldCup/_git/WorldCup c:\course\labs\repos\WorldCup

4. Sign in if/when prompted.

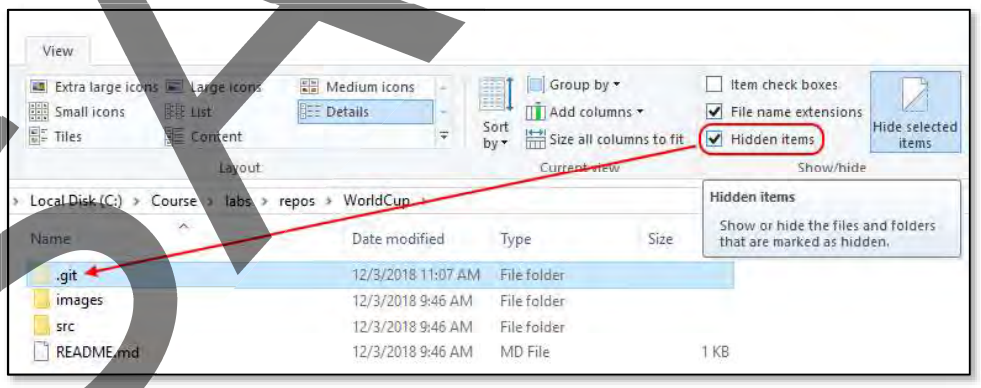
   It shouldn't take long to clone the WorldCup repository.

   ```
   Cloning into 'c:\course\labs\repos\WorldCup'...
   remote: Azure Repos
   remote: Found 27 objects to send. (90 ms)
   Unpacking objects: 100% (27/27), 76.85 KiB | 948.00 KiB/s, done.
   ```

   Cloning is typically a one-time command. Git clone also executes a *git init* command to full initialize the repository, after which, Git for Windows will take over managing the changes. This repository will maintain a connection to the original, remote repository through an *origin*. The origin is a pointer that points you to the original repository that was cloned.

5. Using **File Explorer**, navigate to **C:\Course\Labs\Repos\WorldCup** and explore the newly cloned repo.

6. Show **Hidden items** and inspect the .git folder.



This *.git* directory stores the metadata needed for git to work, including subfolders that contain template, objects, refs, etc. The HEAD files gives the current commit your repo is operating under.

EXERCISE 2 – USE GIT TO TRACK CHANGES

Task: Add, Track, and Commit Changes or

In this task each team member, or pair of team members, will add some folders and files and then commit those changes to the local repository.

1. Open a command window and run the following command to change directory:

   cd c:\course\labs\repos\worldcup

2. Create a new folder and change to that folder:

   md authors
   cd authors

3. Create a short markdown (text file) with your name:

   echo *"<first name> <last name>"* > *<first name>*.md

   Example: *Echo "Richard Hundhausen" > richard.md*

   Note: file names must be unique, so you may have to get creative if your team has duplicate names.

4. Add the newly created file to Git:

   git add *<first name>*.md

   Example: *git add richard.md*

   This command changes the folder into a staging area to let Git know that updates will occur on one or more specific folders. This does not change the directory, and the change will not be made until the *commit* command is executed.

5. Show the current working tree status:

   git status

   Do you see the new file there being tracked? _____

   Git *status* keeps track of the changes done with git *add*. This helps you to know exactly what the Git *commit* command will do. Each file's status can be either *staged*, *unstaged*, or *untracked*. Files can be ignored (untracked) by listing their names or extensions in the .gitignore file.

6. Commit the changes to the local repository:

   git commit -m "Added *<first name>*'s author information"

   Example: *git commit -m "Added Richard's author information"*

Task: Commit and Undo Changes or

In this task each team member, or pair of team members, will commit some more local changes and, realizing you've made a mistake, undo those changes.

1. From the command window, type **Notepad *<first name>*.md**.

   You should be in the *c:\course\labs\repos\worldcup\authors* folder.

2. Update the .md file accordingly …

   ```
   ### <first name> <last name>
   - Email: <email address>
   - Organization: <company name>
   - Nickname: <optional nickname>
   ```

   Example:

   ### Richard Hundhausen
   - Email: richard@accentient.com
   - Organization: Accentient
   - Nickname: rhundhausen

3. Save the file and exit **Notepad**.

4. Commit the changes to the local repository:

   git commit -m "Updated <first name>'s author information"

   Example: *git commit -m "Updated Richard's author information"*

   What message did you receive? _____

   The changes to the file were not staged, therefore Git did not commit the changes to the repository.

5. Confirm this by showing the current working tree status:

   git status

   Is the file "staged" or "not staged" for commit? What color is the message? _____

6. Stage the changes and check the status again:

   ```
   git add <first name>.md
   git status
   ```

   Is the file "staged" or "not staged" for commit? What color is the message? _____

7. Commit the changes to the local repository and check the status again:

   `git commit -m "Updated <first name>'s author information"`

   Example: *git commit -m "Updated Richard's author information"*

   The commit should succeed and the working tree should be clean.

8. Review the full project history:

   `git --no-pager log`

   The *log* command displays committed snapshots, listing the project history. You can filter and search the history. The "--no-pager" switch stops the output from prompting for a page at a time.

   How many entries are listed? _____

9. Show the same history in a condensed, single line format:

   `git log --oneline`

10. Review the Git project history since you cloned the repository:

    `git log origin/main..HEAD`

    This will display commits that occur in a range (e.g. commit IDs, branch names, HEAD, or any other kind of revision reference).

    How many entries are listed? _____

11. From the command window, type:

    `Notepad <first name>.md`

12. "Accidentally" replace the contents of the file with:

    `# 404`

13. Save the file and exit **Notepad**.

14. Stage and commit the changes:

    ```
    git add <first name>.md
    git commit -m "Oops"
    ```

15. After realizing you've wiped out your code, determine the last known good commit:

    `git log --oneline origin/main..HEAD`

    What is the commit ID (SHA-1 hash) of the last commit *before* the Oops commit? _____

    In this example, the commit ID would be 78cc68e …

```
c:\Course\Labs\Repos\WorldCup\authors>git log --oneline origin/main..HEAD
ded6cf4 (HEAD -> main) Oops
78cc68e Updated Richard's author information
389ae8a Added Richard's author information
```

16. Checkout that commit, substituting your commit ID below:

    `git checkout <commit ID>`

    This will put you in a "detached HEAD" state allowing you to review files as they were at this point in time, make experimental changes and even commit them. More importantly, you can't mess up anything here because the HEAD still points to the "Oops" commit.

```
c:\Course\Labs\Repos\WorldCup\authors>git checkout 78cc68e
Note: switching to '78cc68e'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.
```

17. Use **Notepad** to review the contents of **`<first name>.md`**.

    As you can see, your original contents are safe. We will revert back to this good content soon.

18. Close **Notepad**.

19. Undo the previous checkout and move the HEAD back to its original/correct position:

```
git checkout -
git log --oneline origin/main..HEAD
```

20. Revert the **Oops** commit:

    `git revert HEAD`

    This opens *Notepad* allowing you to enter a commit message. Lines starting with "#" are ignored.

21. Close **Notepad** without making any changes.

    The default message will suffice. Upon closing, the revert will occur:

```
[main 5d35098] Revert "Oops"
 1 file changed, 4 insertions(+), 1 deletion(-)
```

22. Review the commit history:

    `git log --oneline origin/main..HEAD`

    Was the *Oops* commit removed from history? _____

    Reverting is a tracked change, so the revert was actually added to the history.

```
c:\Course\Labs\Repos\WorldCup\authors>git log --oneline origin/main..HEAD
5d35098 (HEAD -> main) Revert "Oops"
ded6cf4 Oops
78cc68e Updated Richard's author information
389ae8a Added Richard's author information
```

    Could we revert the revert? Yes. If we ran *git revert head* twice more, here's what the log looks like:

```
c:\Course\Labs\Repos\WorldCup\authors>git log --oneline origin/main..HEAD
1265f81 (HEAD -> main) Revert "Revert "Revert "Oops""
a31c46a Revert "Revert "Oops""
5d35098 Revert "Oops"
ded6cf4 Oops
78cc68e Updated Richard's author information
389ae8a Added Richard's author information
```

Task: Push to Azure Repos or

In this task each team member, or pair of team members, will push their local commit history to the remote repository hosted in Azure DevOps Services.

1. Change to the root of the WorldCup repository:

   ```
   cd c:\course\labs\repos\worldcup
   ```

2. View the connection to your remote WorldCup repository in Azure Repos:

   ```
   git remote -v
   ```

   These repository connections were created automatically when you cloned earlier. Cloning will automatically create a *remote* connection called "origin" pointing back to the cloned repository.

3. Pull any updates from the remote Azure Repos repository to your local repository:

   ```
   git pull
   ```

4. Push your local changes to the remote repository:

   ```
   git push
   ```

   This will enumerate, compress, and write the objects to the remote repository.

   At this point, your local repository has been synchronized with the remote repository.

5. Push your local changes to the remote repository again.

   What message did you receive? _____

6. Coordinate with your colleagues until everyone has *pushed* their own changes and *pulled* everyone else changes.

   At this point, every local repository should have all .md files in the authors folder.

7. Return to the browser and go to the **Files** page on the **Repos** hub.

8. Open the **authors** folder and review the files.

   Do you see everyone's *.md* files? _____

   You may have to refresh the page and/or have them run *git push* first.

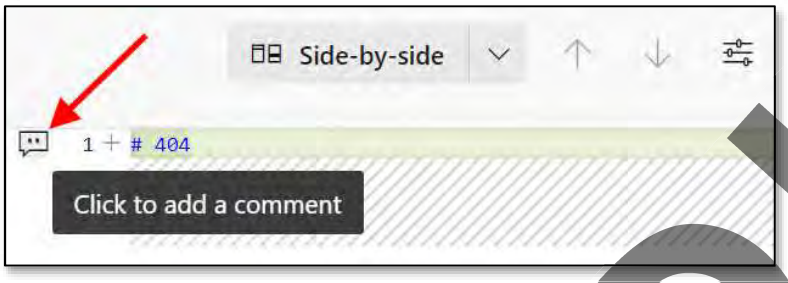
9. Open *your* .md file and view its **History**.

   This shows you a graph of the commits, related messages, authors, and other items of metadata.

10. Click on the **Commit** ID for the **Oops** commit.

11. This shows a side-by-side diff of the Oops commit (on the right) and the previous version (on the left).

12. Add a comment to the # 404 line:



13. Enter **I'll be more careful next time** for the comment.