Consolidating Your Team's Source Code with Version Control

Benjamin Day
TRAINER | COACH | DEVELOPER

@benday www.benday.com

Summary

- Why use version control?
- Two types of TFS version control
- Git basics
- Version control conflicts
- Branching & branch dysfunctions
- Multiple Git repositories in a Team Project
- Code reviews with pull requests

Why Version Control?

Work by yourself

Work with a team

Work with multiple teams

Work by yourself

Life is simple

Protect from your own errors

Backup your code

History

“Diff”

Work with a team

All the same problems as the solo mode

Integration

Combine all work → the usable software

Work with a multiple teams

More people, more complicated

Integrations across teams

- Multiple sub-products?

Combine all work → the usable software

Why use version control?

It is the integration point

Core tools for integrating

Track & compare history

You could do this all
without version control...

...but if you're aspiring to DevOps awesomeness...

...version control is required.

Version control is the core of your DevOps flow.

Next Up:
Two Types of Version Control in TFS

Two Types of Version Control in TFS

Team Foundation Server Version Control (TFVC)

Git

Two Types of Version Control in TFS

Team Foundation Version Control (TFVC)

Created by Microsoft

Available since the beginning of TFS

Team Foundation Server 2005

Centralized Version Control

Git

Created by Linus Torvalds

Initial release in April 2005

Added to TFS in 2013

Distributed Version Control

Centralized vs. Distributed

Centralized Version Control	Distributed Version Control
Needs a network connection	(Mostly) doesn't need a network connection
Add, edit, delete, check in/out, branch, merge	Add, edit, delete, check in, branch, merge, undo available offline
TFS local workspaces → 1 version undo	Revert to a version from 2 years ago?
Ultimately TFS knows what's on your machine	Remote server is completely unaware until you "push"

“Should I use TFVC or Git?”

TFVC: Pros & Cons

Pro	Con
Familiar	Not what the “cool kids” want to use
Changes are easily tracked	Offline support is weak
Granular security controls	TFS code review flow is underwhelming
Handles large repositories with ease	Folder-based branching is hard / annoying
Tooling is mature	

Git: Pros & Cons

Pro

Shiny & New

Offline support is *amazing*

Branching is easier and more understandable

Code reviews with Git (“pull requests”) are really good

Lots of community enthusiasm

Con

Learning curve

Get confused, lose changes

Security is at the repository level

Huge repositories can be challenging & disk intensive

- Microsoft is actively working on this
- Windows team is now using Git
- Largest in the world

Git or TFVC?
No right or wrong answer.

This module and the overall course is going to focus on Git.

TFVC has been stable for years. It hasn't changed much since TFS2012.

Comprehensive Tour of TFS Version Control

"Version Control Basics"

"Version Control Beyond the Basics"

https://app.pluralsight.com/library/courses/alm-fundamentals/table-of-contents

Next up:
Git + TFS

The Basics of Git + TFS

Local vs. Remote

The Basic Git Flow

“Clone” the Remote Repository

- First time only
- Creates local copy

Work Locally

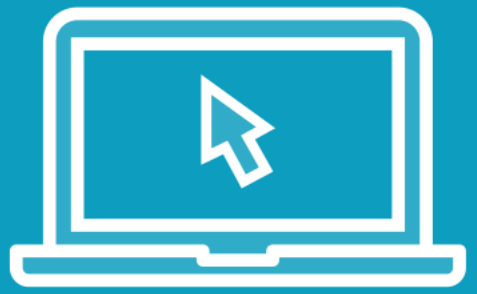
- Add, edit, delete, etc
- “Commit”
- (repeat)

Share

- “Push” to the Remote
- Publishes local commits to remote server

Next up:
Getting started demos for Git & TFS

Demo



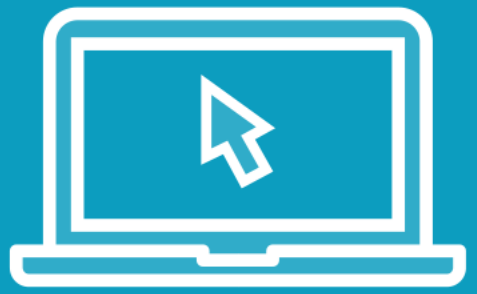
- Create a TFS Team Project
- Initial setup of a Git repository
- Add some code
- Push to TFS
- Web interface for version control

Demo

Git + TFS project management

Associating Work Items to check-ins

Demo



"Get latest" with Git

Fetch, Pull vs. Sync

Next up:
Merge Conflicts in Git

Git Merge Conflicts

Multiple People in Git + Infrequent Connection → Merge Conflicts

Merge Conflicts

Two people change the same piece of code

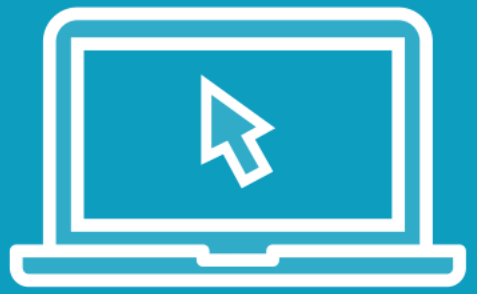
(Happens in TFVC, too)

Longer you wait, more likely you get conflicts

Not the end of the world

Visual Studio has tools to resolve conflicts

Demo



Create a Git conflict between two people

Resolve the conflict

Move on with your lives

Next up:
Multiple Git repos

Demo

Create more Git repos in a Team Project

Favorite Repos

Next up:
Branching & Merging

What is branching & merging?

Branch / Merge

Source control feature

Smart copy

- Branch

Work in isolation

Integrate changes later or never

- Merge

Source control manages relationships

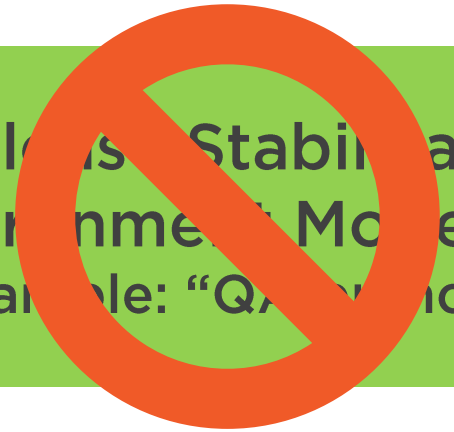
Too many branches →
barrier to done / delivery

When you create a branch, have a good idea for why you're creating that branch.

Four Branching Mindsets

Development Isolation
for Feature or Team

Pre-rel s Stabi ation &
Envi nme Mo eling
(exa le: "Q ch")



Release Maintenance
(aka. Release Snapshot)

Code Reviews

Demo

Create a branch

Merge changes between branches

Next up:
Risks of Branching

Best Practice:
Don’t Branch

Branches can get you in trouble.

Have a good, clear reason
before you create a branch.

A Branch Is an Integration Credit Card

Create as many branches as you want

Code to your heart's content

Eventually, you'll need to do a merge

The merge is your credit card bill

Branching for release is generally safe

Branching for code review...maybe

Branching for development isolation can cause problems

Three ways to get into trouble

1. Branch because it “lets them do more”
2. Communication & trust problems
3. 8,000 branches filled with half-done work

Branching Problem #1:

"It lets us do more"

Too much work in progress

Scrum with 30 day sprints

4 PBIs in current sprint

Team creates 4 feature branches

Merge on the last day

Merges don't go well

Code reviews don't go well

Day 30: Nothing is done → FAIL

Branching Problem #1:

“It lets us do more”

Too many moving parts

Not integrating

Don’t try to do so much stuff

Work on one PBI at a time *AS A TEAM*

Minimize branching

Do frequent “refresh” merges

Refresh Merges

Periodic merges

Ideally: source and target get refreshed

Required: target refreshed from source

- Branched code is refreshed from original

Pay your integration “tax” gradually

Branching Problem #2:

Communication & Trust Issues

More a branching "symptom"

Developers don't trust each other

"Tim's code is awful and breaks all the time so I'm going to work in a branch."

Result: integration problems

Answer:

- Talk to one another!
- Fix the *actual* problem

Branching Problem #3:

Unclear Goals, Shifting Goals

~20,000 branches

- Lots of partially done work
- Unclear what's important

Usually a management problem

No clear goals from management

Management changes goals constantly

Constant production fixes

Answer: Don't do this.

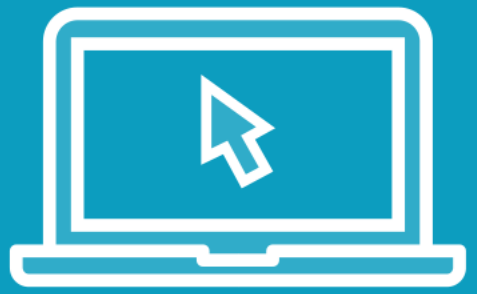
- Task switching isn't free

Best Practice:
Don’t Branch

If you must branch,
have a good reason &
integrate often.

Next up:
Code Reviews using Pull Requests

Demo



- Create a branch
- Request a merge using a “pull request”
- Review code changes
- Squash changes

Summary

- Why use version control?
- Two types of TFS version control
- Git basics
- Version control conflicts
- Branching & branch dysfunctions
- Multiple Git repositories in a Team Project
- Code reviews with pull requests

Next up:
DevOps Mindset &
DevOps Metrics