Migrating to Git and Staying Alive

Hello!

I am Leon Jalfon

DevOps & ALM Architect
at Sela Group

leonj@sela.co.il
http://blogs.microsoft.co.il/leonj
http://sela.co.il/

Agenda

- Introduction
- The Reasons
- The Implications
- The Case Study
- The Plan
- The Migration
- The Transition
- Conclusions

1

Introduction

TFVC and Git Comparision

Workflow

TFVC is composed by incremental changesets

Git use SHA's to store changes (commits)

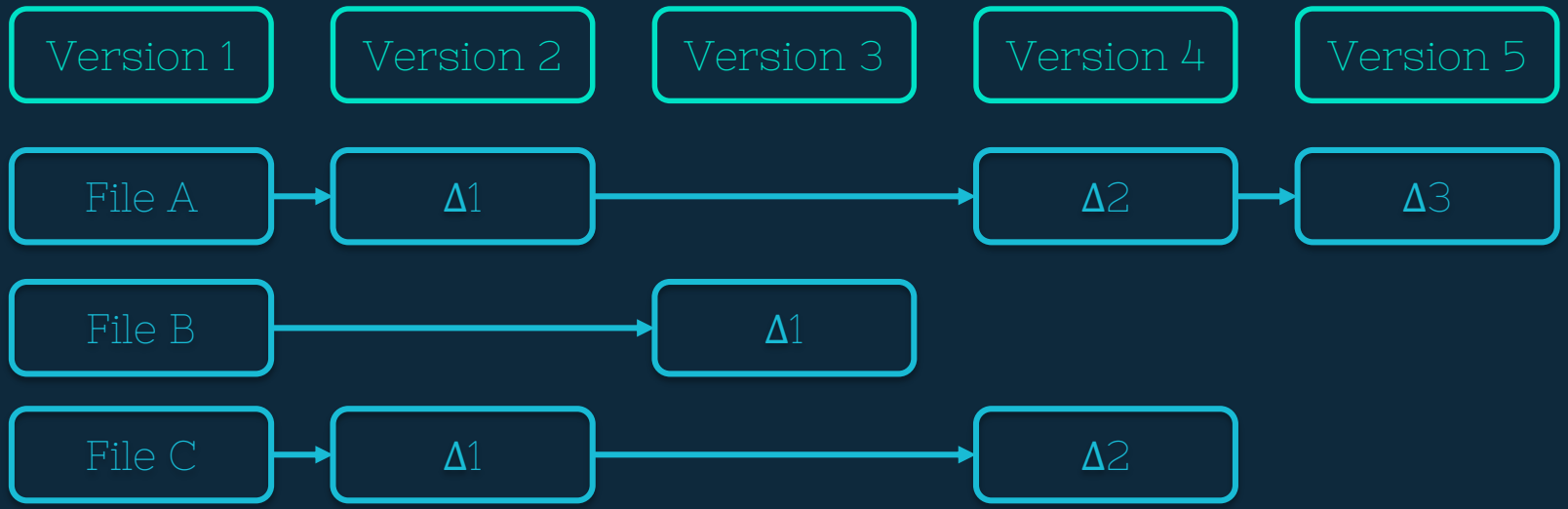
Which changes are tracked

TFVC track changes, changesets are deltas of the changes to the content

Git is all about content, Git commits are snapshots of the content

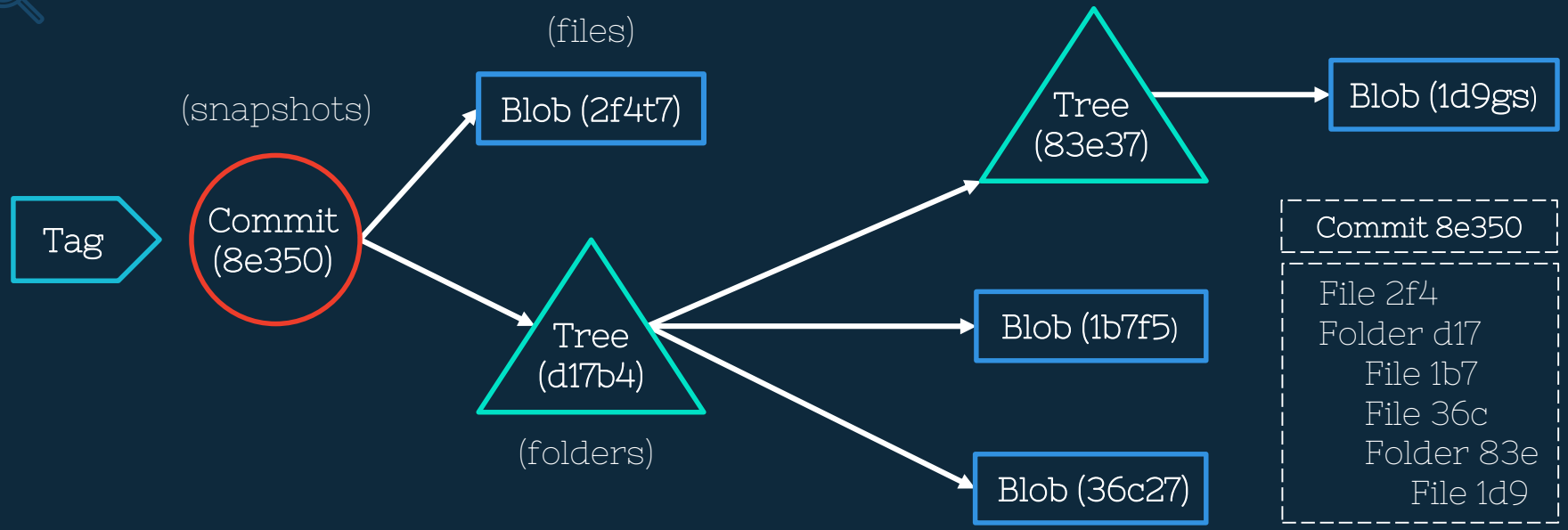
How changes are stored

TFVC, Incremental (deltas)



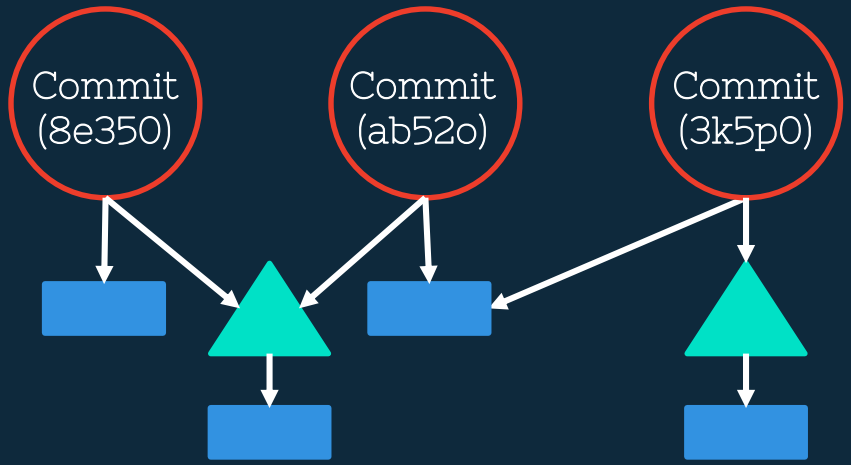
How changes are stored

Git, Snapshots



How changes are stored

Git, Snapshots



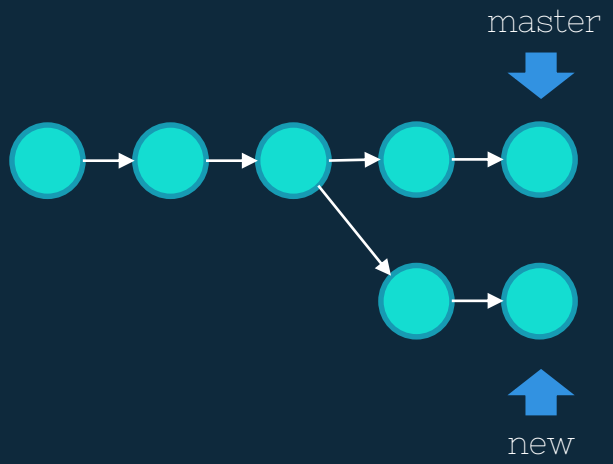
Branching

TFVC: Branches are folders

GIT: Branches are pointers

(main)
File 1
Folder
File 2

(new)
File 1
Folder
File 2



Workspace Vs Clone

TFVC: Mapping of server paths locally

GIT: Whole repository in the .git folder

$/Path/BranchA	C:/Path/BranchA
$/Path/BranchB	C:/Path/BranchB
$/Path/BranchC	C:/Path/BranchC
$/Path/BranchD	C:/Path/BranchD

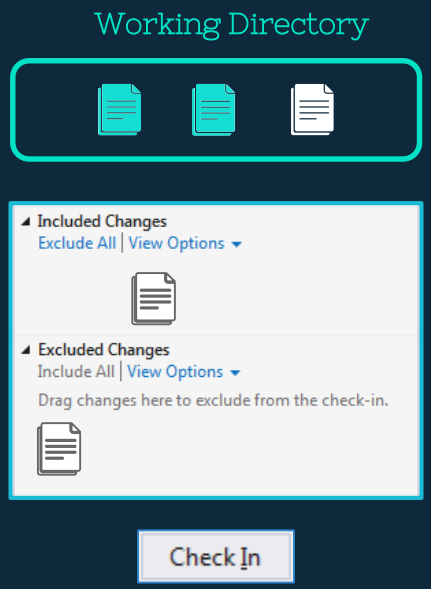
git

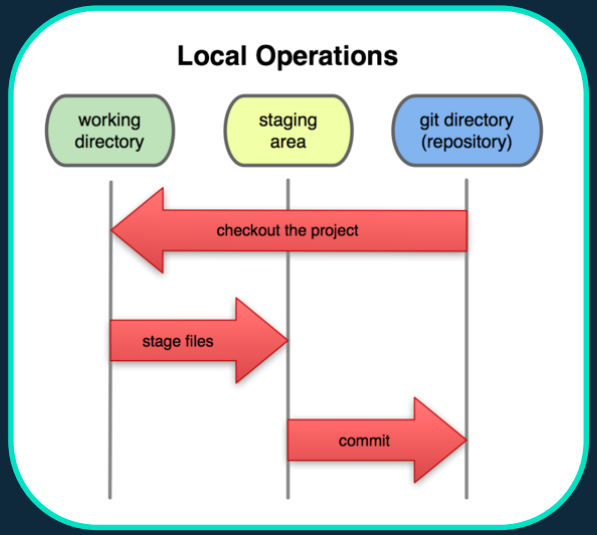
.git

Staging Files

TFVC: Exclude include pending changes

GIT: Add, Commit and Push

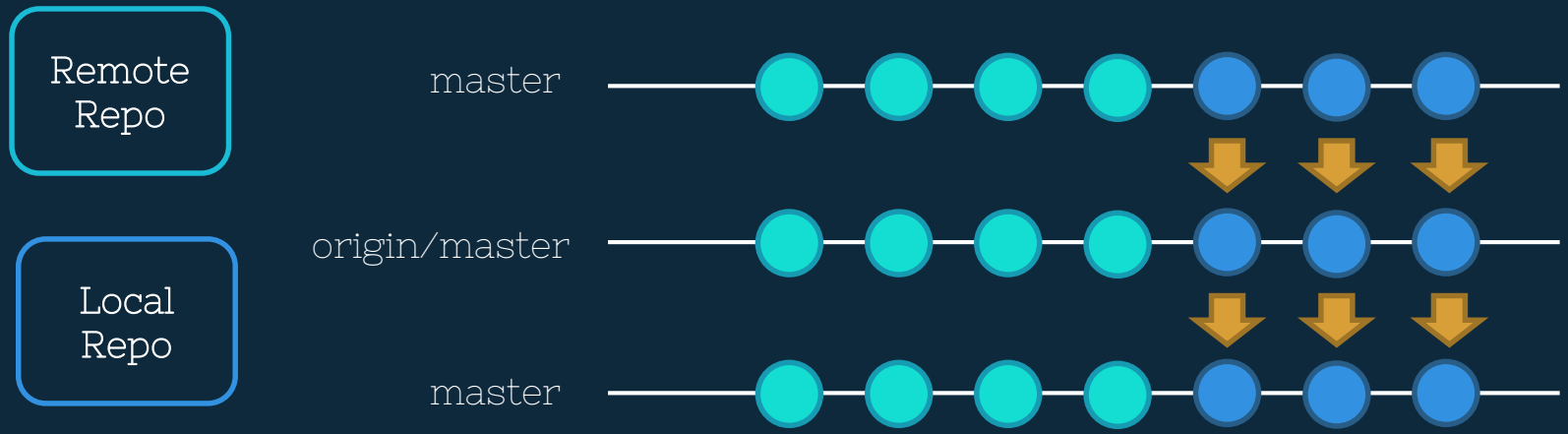




Get Latest Version

TFVC: update working directory

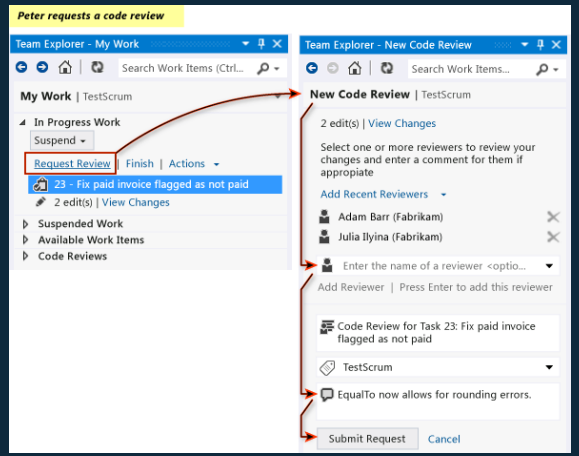
GIT: Remotes branches to synchronize



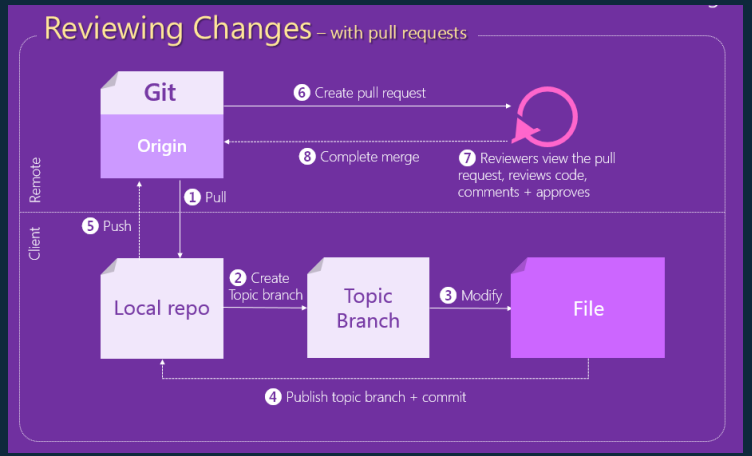
Code Review

TFVC: Using shelvesets

GIT: Pull Requests

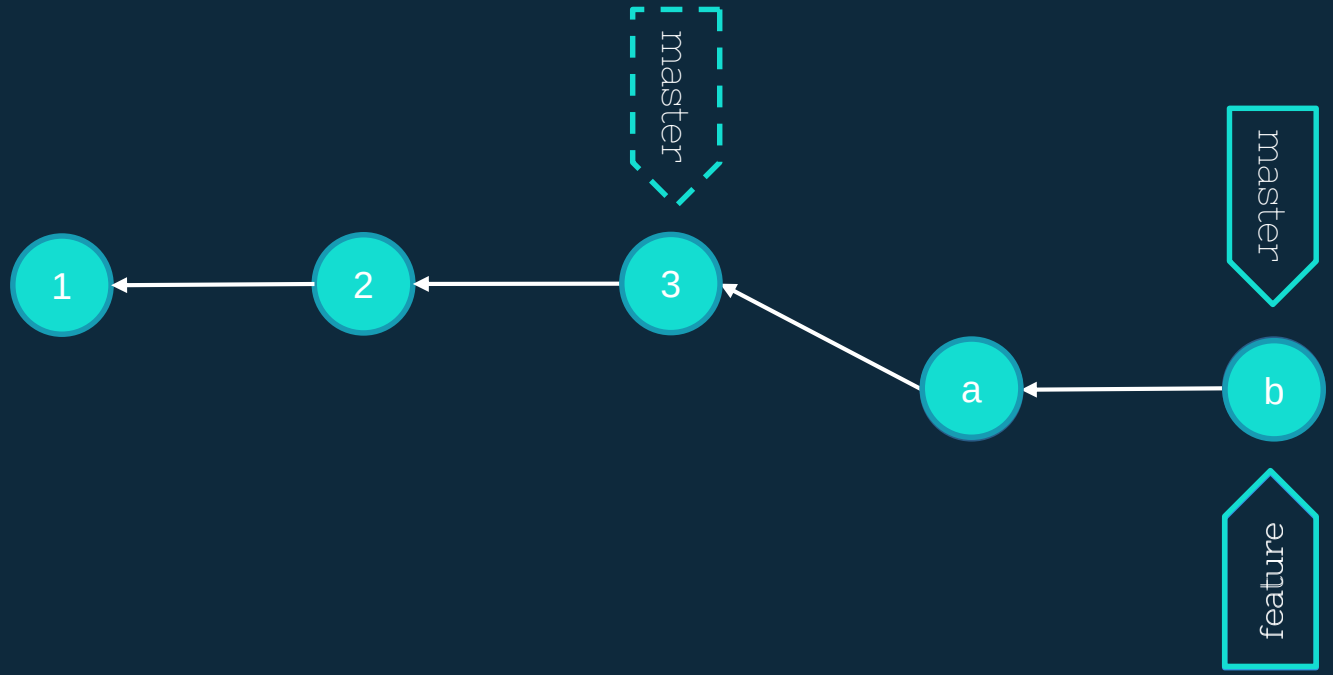


If you change the code you need to create a new shelveset



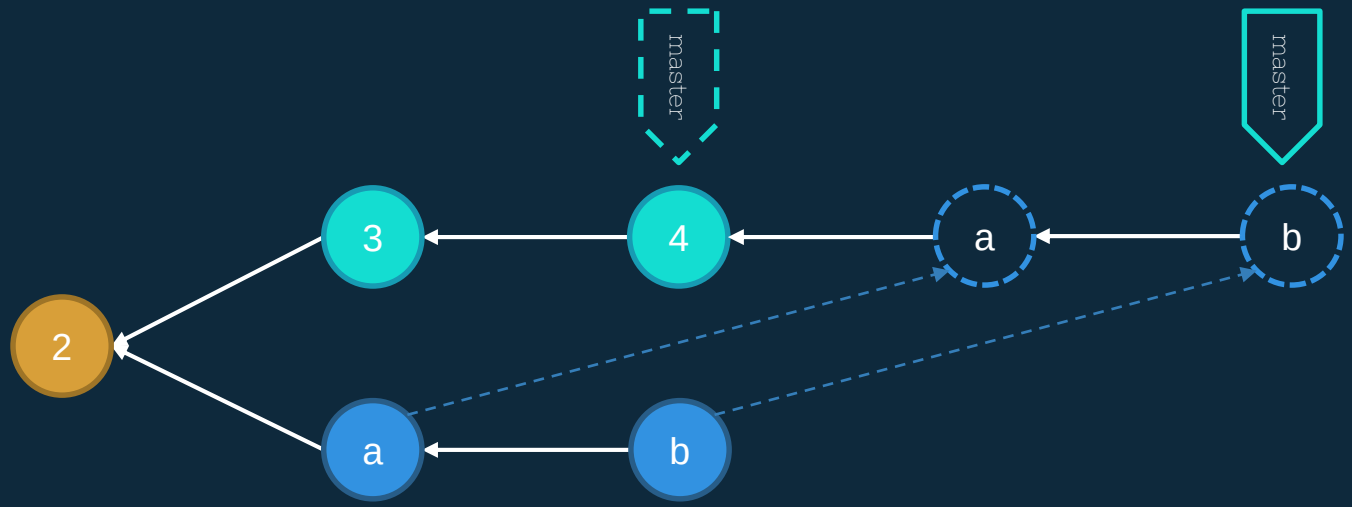
Updated automatically, you can add conditions, conflicts resolved before merge

Fast-Forward Merge



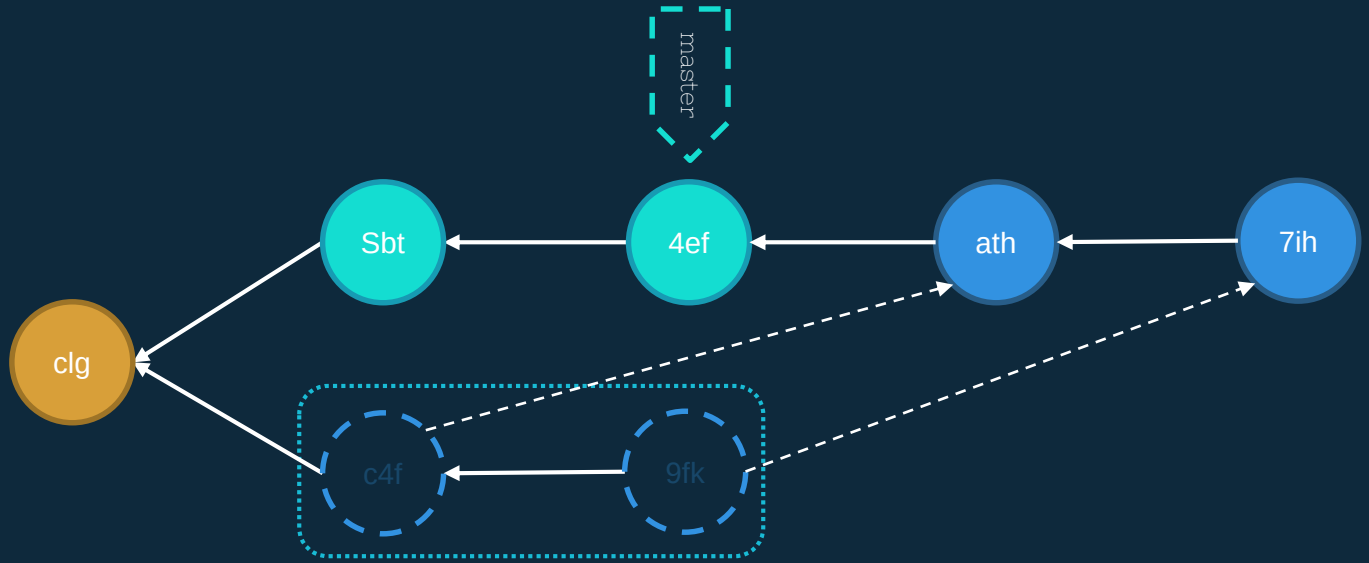
Rebasing

How it Looks...



Rebasing

What really happens…



$ git gc

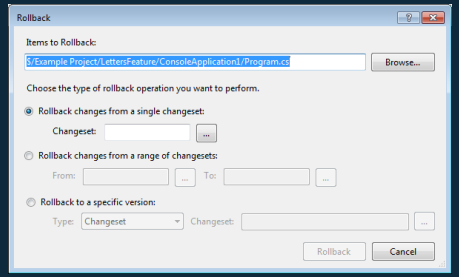
Undoing Changes

TFVC: rollback

GIT: Revert, reset and cherry-pick

Rollback:

- From a single changeset
- From a range of changesets
- To a specific version



Checkout

- Check out a previous version of a file

Revert

- Generate a new commit that undoes all of the changes introduced in specific commit

Reset

- Move the commit reference

Cherry-Pick

- Generate a new commit that include the changes introduced in the specified commit

Label Vs Tags

Label (TFVC): Mutable grouping of specific version of a set of source files

Tags (GIT): Is a named pointer to a commit in the repository. Useful for making a point in time in your repository.

Shelveset Vs Stash

Shelveset (TFVC): Is a set of pending changes are temporarily saved on the server

Stash (GIT): Is a set of pending changes are temporarily saved in the local repository

2

The Reasons

Why and when to migrate? (or not)

Why Migrate to Git?

- Is the "de facto" standard for version control system
- There is more online support
- Is Cross-platform
- Is free and open source
- Hosting Freedom
- Small size (much better compression)

Why Migrate to Git?

- Distributed and decentralized (work offline)
- Git is fast, really fast (Local Operations)
- Flexibility (simple to migrate to another place)
- Security (implicit backups, unique SHA's)
- Integration with another tools

Why Migrate to Git?

- Better for distributed teams
- Local history (offline)
- Local commits and local branching
- Frequent commits (and deliver the code only when ready)
- Branching and merging more simple and efficient
- Rewrite history (rebase)

Why Migrate to Git?

- Less merge conflicts (three-way merge)
- Fast-forward merging
- Better code review system (pull requests)
- Custom workflows (git flow)
- More operations (Git blame, git bisect)
- Multiple Git repositories in a single Team Project (in TFS)
- Microsoft is using Git for version control

Why continue using TFVC?

- Git is hard to start with (to work with Git, you have to understand how Git works)
- Bad binaries handling (Git LFS)
- Bad performance in big repositories (recommended < 1GB)
- Bad for long history (> 50.000 commits)

Why continue using TFVC?

- No gated checkins (feature branches)
- Crazy command line syntax and crappy documentation:
  - **git push** – Update remote refs along with associated objects
  - **git rebase** – Forward-port local commits to the updated upstream head

Why continue using TFVC?

- Simple tasks need so many commands (add, commit, push, pull request)
- Responsibility for maintaining the repository is passed to the contributors (In TFVC, only one person had to deal with the complexities of branches and merges)

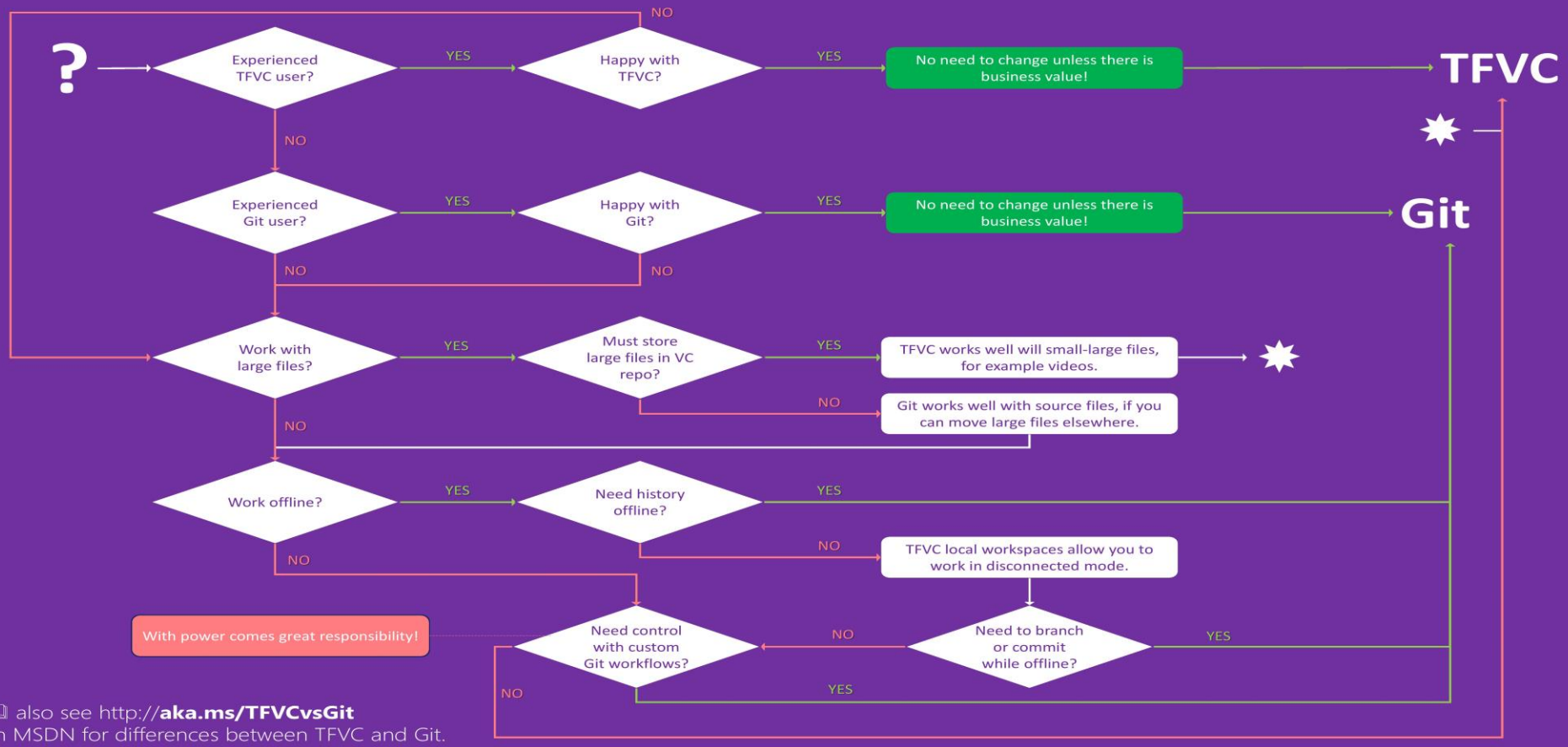
Why continue using TFVC?

- Unsafe version control (git reflog)

So when to migrate and when not?

Version Control consideration aid for TFVC vs. Git

Visual Studio
ALM Rangers



📖 also see http://**aka.ms/TFVCvsGit** on MSDN for differences between TFVC and Git.

3

The Implications

Things to take into account, consequences of migrating and common pitfalls

To Take Into Account

- A good training is elementary
  - Developers
  - Administrators
- Migrating to Git entails a new workflow
- Git is learned by using it
- Code Review process changes
- The migration tools are not perfect (you should perform good testing to ensure the success of the process)

To Take Into Account

- The following things must be adapted
  - Builds (good opportunity to update)
  - Tests (and automation)
  - Source control integrations
  - TFS plugins (related to SCM)
  - Release management
- Migrate to Git is much more than just migrating the code

Common Pitfalls

- Bad training
- No support after migration
- Wrong workflow design
- A lot of binaries in the repository
- Hurry up (to go fast, row slowly)
- Bad adaptation, try to use Git as if it was TFVC (instead of learning to “think” in Git)

Tips & Recommendations

- Define the migration objectives from the beginning
- Give the migration the importance it deserves, don’t take it lightly
- The less history we migrate the better
- If it’s possible, start with a "pilot" project
- Print cheat sheets from the new workflow (for the transition process)

4

The Case Study

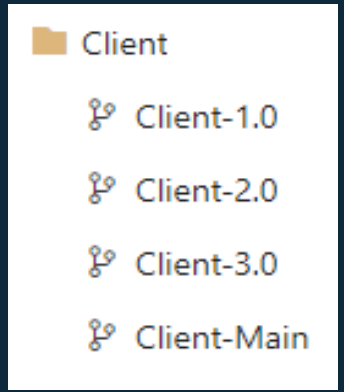
Case study for the demo

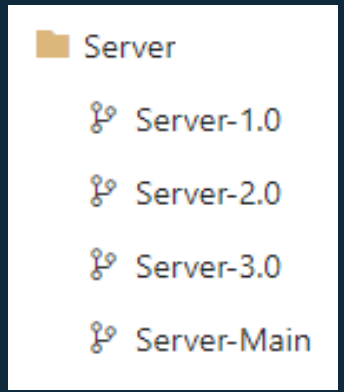
Description

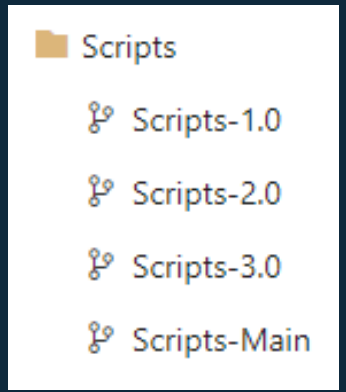
- Pet Shop is a e-Commerce website
- It's composed by 3 projects
  - Client
  - Server
  - Scripts
- All the projects are stored in separate folders under the same TFVC repository

The Source Code

- Each project have 4 identical branches







The Builds

- There are one build per release

Mine | All Definitions | Queued | XAML

Requested by me	Status	Triggered by
PetShop-3.0 : #20170903.2 leon jalfon hotmail requested 5 minutes ago	succeeded	Branched from $... 105 in $/M...
PetShop-Main : #20170903.3 leon jalfon hotmail requested 2 minutes ago	succeeded	add create sum... 106 in $/M...
PetShop-2.0 : #20170903.2 leon jalfon hotmail requested 6 minutes ago	succeeded	Branched from $... 103 in $/M...
PetShop-1.0 : #20170903.2 leon jalfon hotmail requested 7 minutes ago	succeeded	Branched from $... 101 in $/M...

The Situation

- The directive wants to release the project as open source
- Many developers are using Linux
- The team is located in multiple locations and need to work offline
- Managers want to improve the code review process using pull requests

5

The Plan

How to plan a successful migration, best practices and migration tools

The Plan

1. Confirm that there is a business value
2. Create the team that will be in charge of the migration
3. Get ready with Git or seek for professional advice
4. Define the new workflow and branch strategy
5. List the things that need to be updated
6. Define what is to be migrated and where

The Plan

7. Decide what to do with the binaries (delete/keep/migrate to LFS)
8. Restructure permissions (they are different in Git)
9. Define what changes to make in TFVC to facilitate migration
10. Define the migration objectives
11. Decide which migration tool to use
12. Assign roles and set due dates

The Plan

13. Prepare training (for users and admins)
14. Perform tests and document the steps
15. Define and announce the migration final date (downtime)
16. Arrange a post-migration support team
17. Follow-up the team to ensure a successful transition

Let’s Migrate to Git!

Let's see the whole thing in action

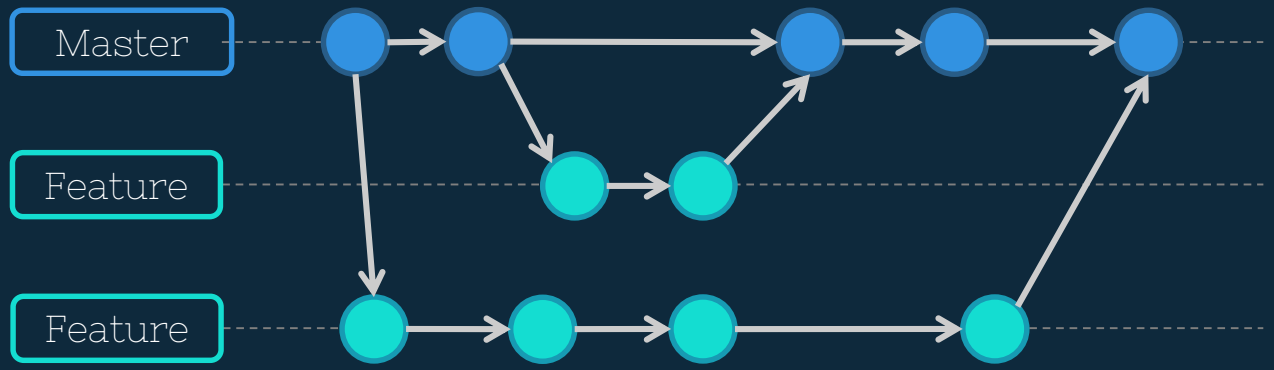
1. Confirm that there is a business value

- The directive wants to release the project as open source
- Many developers are using Linux
- The team is located in multiple locations and need to work offline
- Managers want to improve the code review process using pull requests

2. Create the team that will be in charge of the migration
   - The ALM team will lead the operation together with the DevOps team and the R&D managers
3. Get ready with Git or seek for professional advice
   - The company have Git experts which can plan and perform the migration but we will ask “Sela Group” to direct the training

4. Define the new workflow and branch strategy

- There will be a master branch and “feature branches” will be created for each feature



- Tags will be used to mark the releases

5. List the things that need to be updated
   - Builds
   - Plugins

6. Define what is to be migrated and where
   - We want migrate the code to 2 different repos:
     - petshop-app
     - petshop-scripts
   - Repositories will be hosted in VSTS and GitHub

petshop-script Repository

- Migrate only the last revision of the Main branch to the master branch
- No need to keep work items links
- Remove the git-tfs metadata

MINGW64:/c/Users/leonj/Desktop/ug-demo/petshop-client

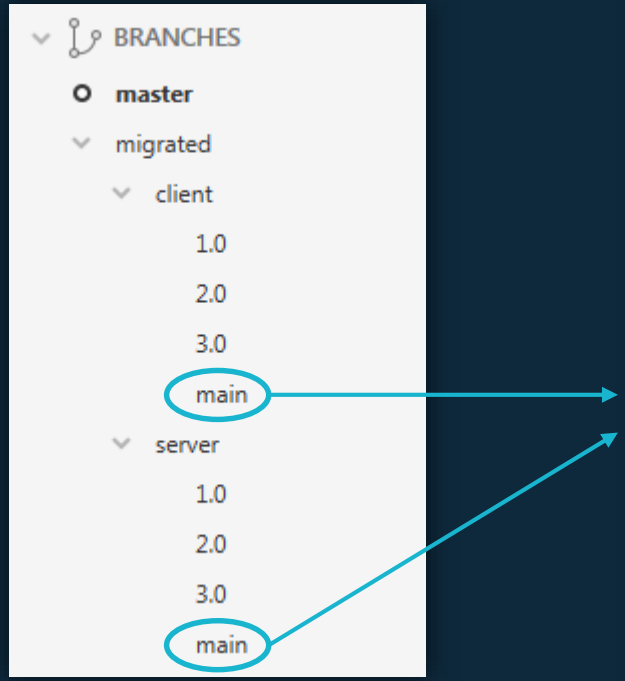
```
commit 58045173457fcbbc6590135b65563f3ae2ffe825 (HEAD -> master, tfs/default)
Author: leon jalfon hotmail <leonjalfon1@hotmail.com>
Date:   Sun Sep 3 19:46:15 2017 +0000

    create blog

    work-items: #1773
    git-tfs-id: [https://leonj.visualstudio.com/DefaultCollection]$/Migration-Demo/Client/Client-Main;C91

Notes:
    Workitems:
    [1773] Create Blog
        https://leonj.visualstudio.com/DefaultCollection/WorkItemTracking/WorkItem.aspx?artifactMoniker=1773
:
```

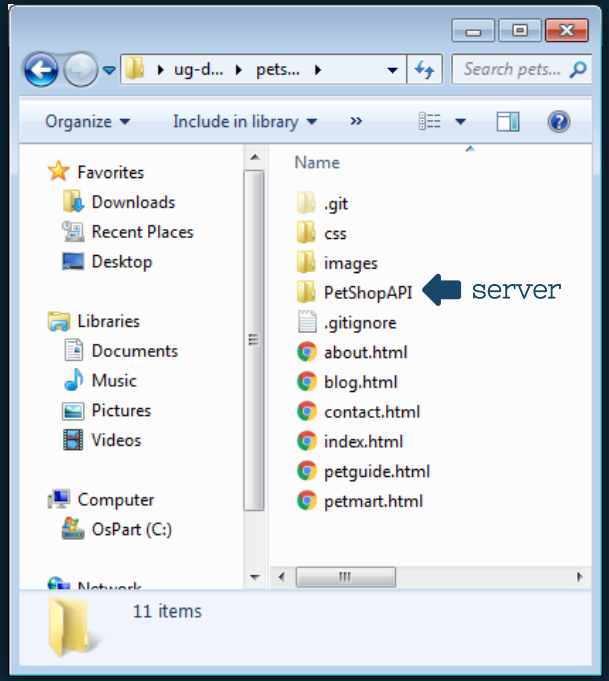
petshop-app Repository

- Migrate the whole history
- Keep work items links
- Add .gitignore file
- Master will contain the merge of the last version of each project
- Use the ref path “migrated/<project>/<branch” to organize the migrated branches
- Change the git-tfs metadata to human format

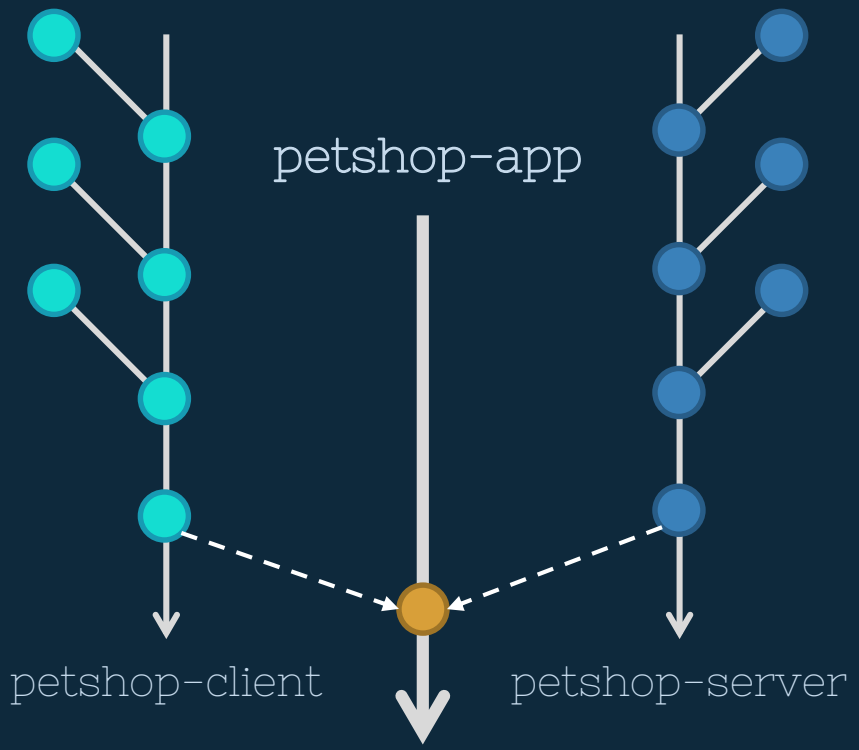
petshop-app

Master Branch





◇ petshop-app



7. Decide what to do with the binaries (delete/keep/migrate to LFS)
   - Executables will be removed (.exe and .jar)
   - Assets .jpg will be migrated to Git LFS
   - Assets .gif will be kept in the repository (rarely edited)

8. Restructure permissions (they are different in Git)
   - Managers will receive repo admin permissions
   - Only DevOps should have access to scripts repo

9. Define what changes to make in TFVC to facilitate migration
   - Every thing is ready, we will organize the repository once everything has been migrated

10. Define the migration objectives
    - Migrate the main branches to 2 new repos
    - Provide a good Git training to all R&D
    - Migrate the work item links to changesets
    - Publish the project in GitHub as open source

11. Decide which migration tool to use

	tf-git	git-tfs
Cross platform	Yes	Only for Windows
Performance	notably slower	nice performance
Branches	Copy the whole repository as a single folder	can try to map your TFS branches to your git branches
TFS checkins	have a option to make each commit a checkin in TFS	have a nice checkin-tool
Disk Space	works by populating the git repository directly	works by creating a TFS working folder mapping in a hidden folder
Development	has not had the community adoption	is actively developed by a community
Lifecycle	has reached end-of-life	Currently mainteined

12. Assign roles and set due dates

- Build: DevOps Team
- Code Migration: ALM Team
- Due date: 01/09/17

13. Prepare training (for users and admins)

- Send developers and admins to take a Git course in a specialized place (by groups)
- Give developers and admins good tools for self-learning

14. Perform tests and document the steps

- Perform the code migration and write a “step by step guide”
- Test that all integrations are working using the migrated repository

15. Define and announce the migration final date (downtime)

- Migration Date: 06/09/17 – 19:00
- Downtime: from 18:00 until 22:00

16. Arrange a post-migration support team
    - Create support team (experimented users)

17. Follow-up the team to ensure a successful transition
    - Print cheat sheets for the new workflow
    - Print cheat sheets mapping the old TFVC functions to Git
    - Ask for feedbacks
    - Monitor repository to discard misuses

6

The Migration

How to perform the migration (demo)

Map Branches

```
git tfs list-remote-branches
http://tfs:8080/tfs/DefaultCollection
```

Migrate a Single Branch

```
git tfs clone
http://Server:8080/tfs/Collection
$/Project/Folder/Branch
Target/Path
```

Migrate Branch and it's Sub-branches with Work Items Links

```
git tfs clone
http://tfs:8080/tfs/DefaultCollection
$/Project/Folder/Branch
--branches=all
--export
```

Migrate Only Last Version

```
git tfs quick-clone
http://tfs:8080/tfs/DefaultCollection
$/Project/Folder/Branch
```

Migrate Specific Changeset

```
git tfs quick-clone
http://tfs:8080/tfs/DefaultCollection
$/Project/Folder/Branch
-c=<ChangesetId>
```

Remove git-tfs Metadata

```
git filter-branch -f --msg-filter "sed
's/^git-tfs-id:.*$//g'" -- --all
```

Update git-tfs Metadata to Readable Format

```
git filter-branch -f --msg-filter "sed
's/^git-tfs-id:.*;C ([0-
9]* )$/Changeset: 1/g'" -- --all
```

Clean git-tfs Workspace

```
git tfs cleanup
```

Clean Git Repository

```
git reflog expire --expire=now --all &&
git gc --prune=now --aggressive
```

Remove Files From The Repository History

(using git BFG)

```
java -jar "path/to/bfg.jar"
--delete-files "*.{exe}"
--no-blob-protection
"path/to/<Repository>"
```

Convert Files to Git LFS

(Using git-lfs-migrate)

```
java -jar “path/to/git-lfs-migrate.jar"
-s “path/to/<Repository>"
-d “path/to/<Converted-Repository>“
"*.gif" "*.jpg" "*.tar" "*.exe" "*.jar" "*.zip"
```

7

The Transition

How to adjust to Git after the migration

Help your Team

- Be aware you will receive many questions the first few weeks after the migration
- Put your old TFVC repository in read only mode to avoid confusions
- Do not answer specific questions, teach them to think in Git
- Give cheat sheets to the developers (new workflow and Git commands)

Git Cheat Sheet

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository
$ git init [project name]

Download from an existing repository
$ git clone my_url

Observe your Repository

List new or modified files not yet committed
$ git status

Show the changes to files not yet staged
$ git diff

Show the changes to staged files
$ git diff --cached

Show all staged and unstaged file changes
$ git diff HEAD

Show the changes between two commit ids
$ git diff commit1 commit2

List the change dates and authors for a file
$ git blame [file]

Show the file changes for a commit id and/or file
$ git show [commit]:[file]

Show full change history
$ git log

Show change history for file/directory including diffs
$ git log -p [file/directory]

Working with Branches

List all local branches
$ git branch

List all branches, local and remote
$ git branch -av

Switch to a branch, my_branch, and update working directory
$ git checkout my_branch

Create a new branch called new_branch
$ git branch new_branch

Delete the branch called my_branch
$ git branch -d my_branch

Merge branch_a into branch_b
$ git checkout branch_b
$ git merge branch_a

Tag the current commit
$ git tag my_tag

Make a change

Stages the file, ready for commit
$ git add [file]

Stage all changed files, ready for commit
$ git add .

Commit all staged files to versioned history
$ git commit -m "commit message"

Commit all your tracked files to versioned history
$git commit -am "commit message"

Unstages file, keeping the file changes
$ git reset [file]

Revert everything to the last commit
$ git reset --hard

Synchronize

Get the latest changes from origin (no merge)
$ git fetch

Fetch the latest changes from origin and merge
$ git pull

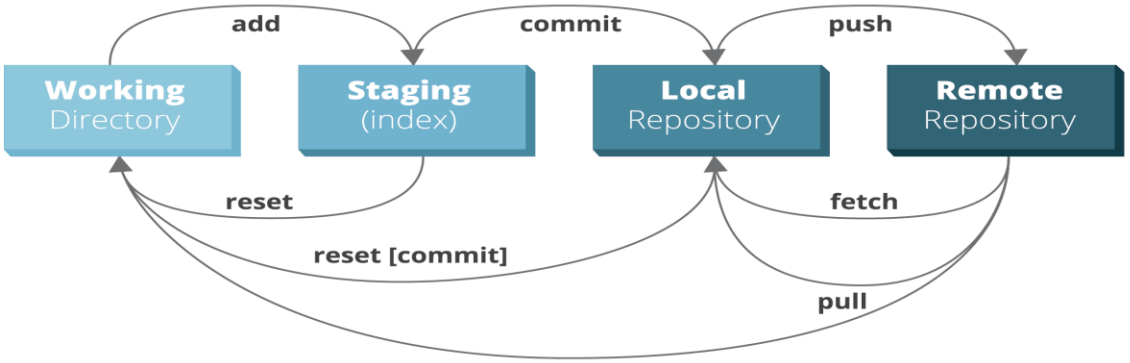
Fetch the latest changes from origin and rebase
$ git pull --rebase

Push local changes to the origin
$ git push

Finally!

When in doubt, use git help
$ git command --help

Or visit https://training.github.com/ for official GitHub training.

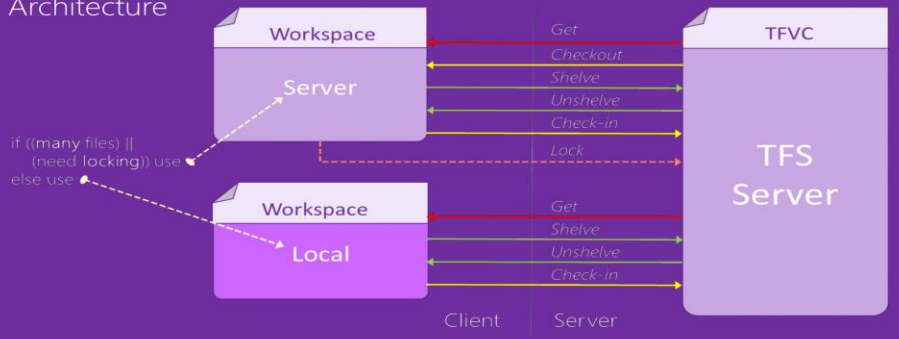


Version Control Cheat Sheet for TFVC and Git

Visual Studio
ALM Rangers

TFVC

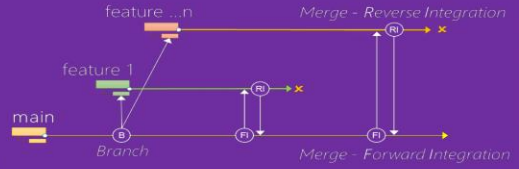
Architecture



Concepts

- **Branch** is an isolated copy of item metadata and version control history.
- **Changeset** is a logical container for changes of a single check-in.
- **Check-in** commits pending changes in workspace to server as a changeset.
- **Label** mutable grouping of specific version of a set of source files.
- **Shelveset** is a set of pending changes are temporarily saved on the server.
- **Workspace** is a local copy of your team's codebase. Create multiple workspaces and switch among them to work on different branches or copies of the codebase.

Feature Isolation Branching Example



Commands

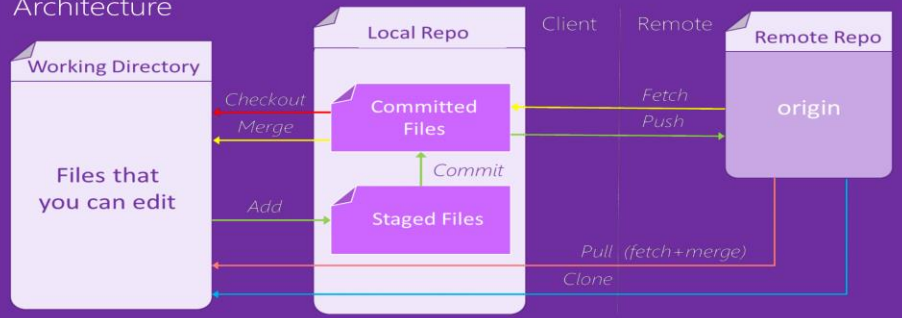
TF *command*
Team Foundation Version Control (TVFC) command line tool with a variety of commands.

- everyday
  - **add** adds new files and folders from a local file system location
  - **get** retrieves a copy of items from TFVC to the workspace
  - **checkin** commits pending changes in current workspace to TFVC
  - **checkout** makes local file writable and changes status to "edit"
  - **delete** removes files and folders from TFVC and disk
  - **history** displays the revision history for files and folders
- branching
  - **branch** creates a copy of items, preserving a relationship to the original items
  - **merge** applies changes from one branch into another
- shelving
  - **shelve** stores a set of pending changes in TFVC without a commit
  - **shelvesets** used to view details of a shelveset or view shelvesets belonging to a specific use
  - **unshelve** restores shelved changes from TFVC to current workspace
- uncommon
  - **changeset** displays information about a changeset
  - **destroy** permanently deletes, version-controlled files from TFV. Admin only!
  - **lock** locks or unlocks to prevent against checkin/checkout of items
  - **rename** changes the name or the path of items
  - **rollback** reverts the changes of one or more changesets
  - **undelete** restores items that were previously deleted
  - **workspace** creates, deletes, displays, or modifies properties and mappings associated with a workspace.
- help
  Type **TF /?** on the command line for a complete list of commands and arguments.

TFSDeleteProject
Command line tool which deletes a team project from a TFS team project collection. It is a non-recoverable operation!

Git

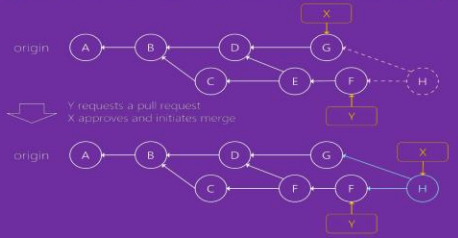
Architecture



Concepts

- **Branch** is a named pointer to a commit history in the repository. Used to isolate changes from each other.
- **Commit** (noun) is equivalent (mostly) to Changeset, but can also be an action.
- **HEAD** is a pointer to the branch your commits will be associated with.
- **Stash** is a set of pending changes are temporarily saved in the repository.
- **Tag** is a named pointer to a commit in the repository. Useful for marking a point-in-time in your repository.

Topic Branch & Pull Request Example



Links

- msysgit.github.io ... Windows client downloads
- git-scm.com ... documentation
- github.com ... code host
- bit.ly/gitscc ... Git source control provider VS2008-2012
- bit.ly/gitext ... Git extensions

Commands

git *command*
git command line tool with a variety of commands.

- everyday
  - **add** adds the current content to existing paths
  - **clone** create a working copy from existing repository
  - **commit** (verb) all the staged local changes
  - **fetch** the latest changes from origin, not merging
  - **pull** the latest changes from origin and merge into working copies
  - **push** commits changes to origin
  - **rm** removes files from the working tree and from the index
  - **status** shows uncommitted changes in the working directory
  - **tag** mark a version or milestone
- branching
  - **branch** [name] creates branch called *name* based on HEAD
  - **branch** [–d name] deletes branch called *name*
  - **checkout** [id] switch to the *id* branch
  - **diff** shows changes to tracked files
  - **merge** two branches
- uncommon
  - **blame** show who changed what and when
  - **bisect** find regressions
  - **grep** search working directory
  - **log** show history of changes
  - **show** [id:file] show a specific *file* from a specific *id*
- help
  Type **git -help** on the command line for a complete list of commands and arguments.

8

Conclusions

Overall summary

Conclusions

- Migrate to Git is much more than just migrating the code
- Migrate to Git entails a new workflow
- A good training is elementary
- Perform good testing to ensure the success of the process

Conclusions

- Migrating the whole story doesn't have to be the best option
- If you plan well, there is no reason to go wrong
- Git and TFS are very different control versions (learn to think in Git!)
- Remember, no need to change unless there is business value!

Thank you for coming!

Any questions?

You can find me at:

- leonj@sela.co.il
- http://blogs.microsoft.co.il/leonj