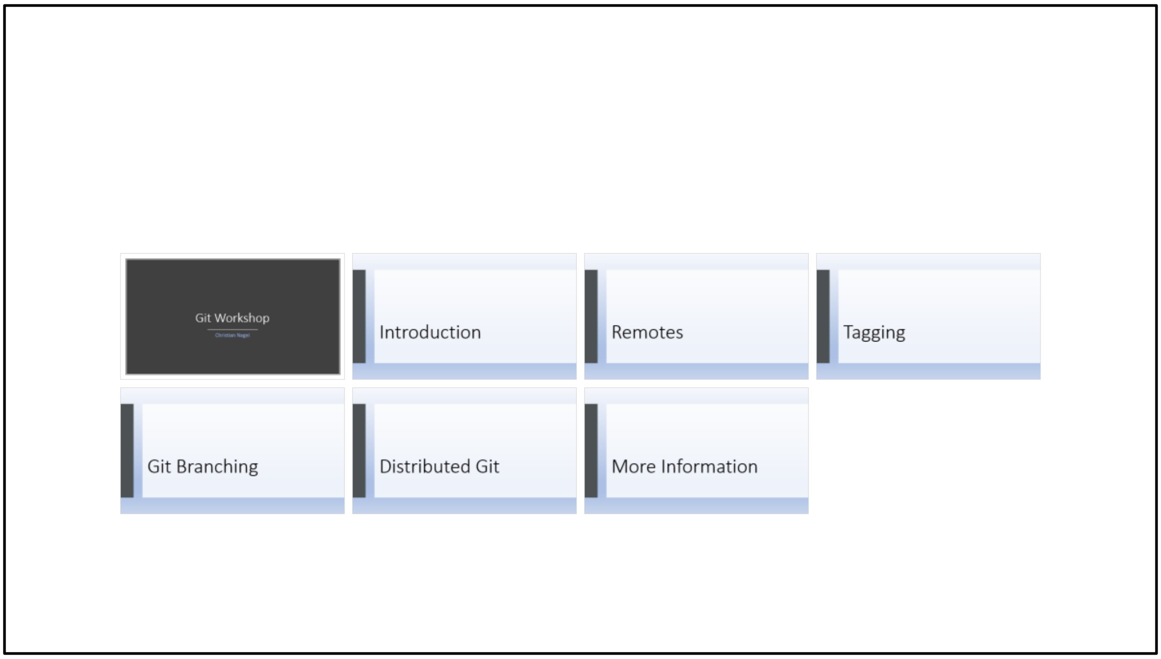


Git Workshop

Christian Nagel

Introduction

Remotes

Tagging

Git Branching

Distributed Git

More Information

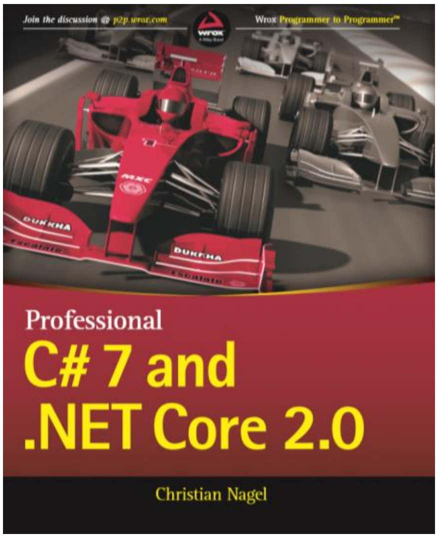
Git Workshop

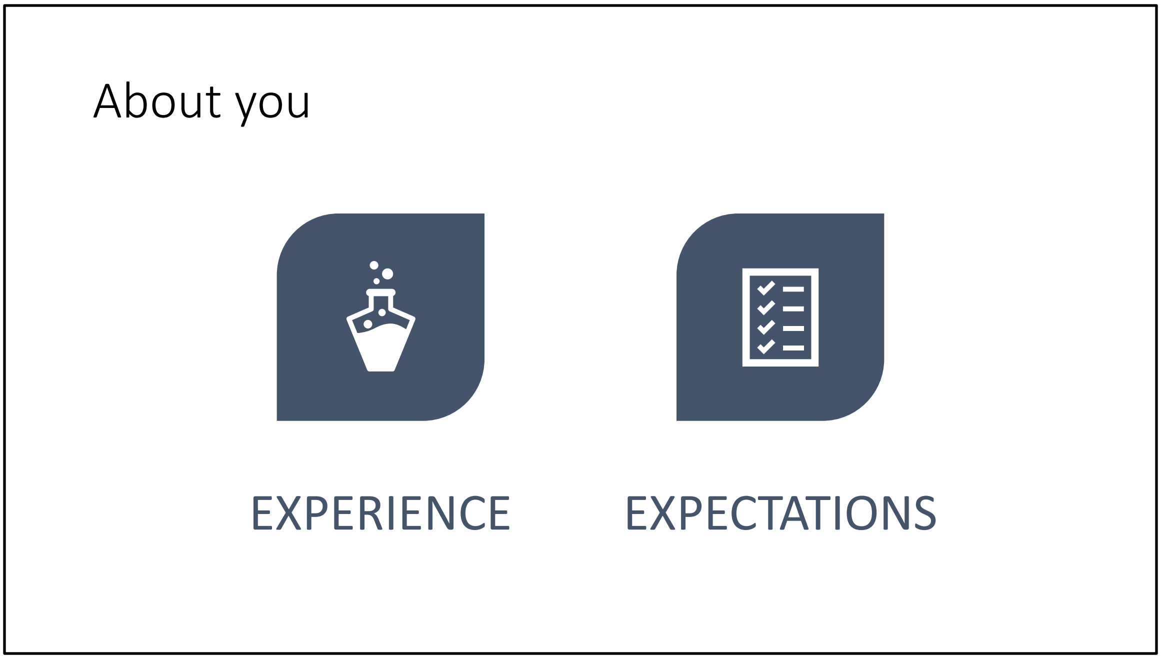
Christian Nagel

Christian Nagel

- Training
- Coaching
- Coding
- Writing

- csharp.christiannagel.com
- www.cninnovation.com
- @christiannagel
- Microsoft MVP

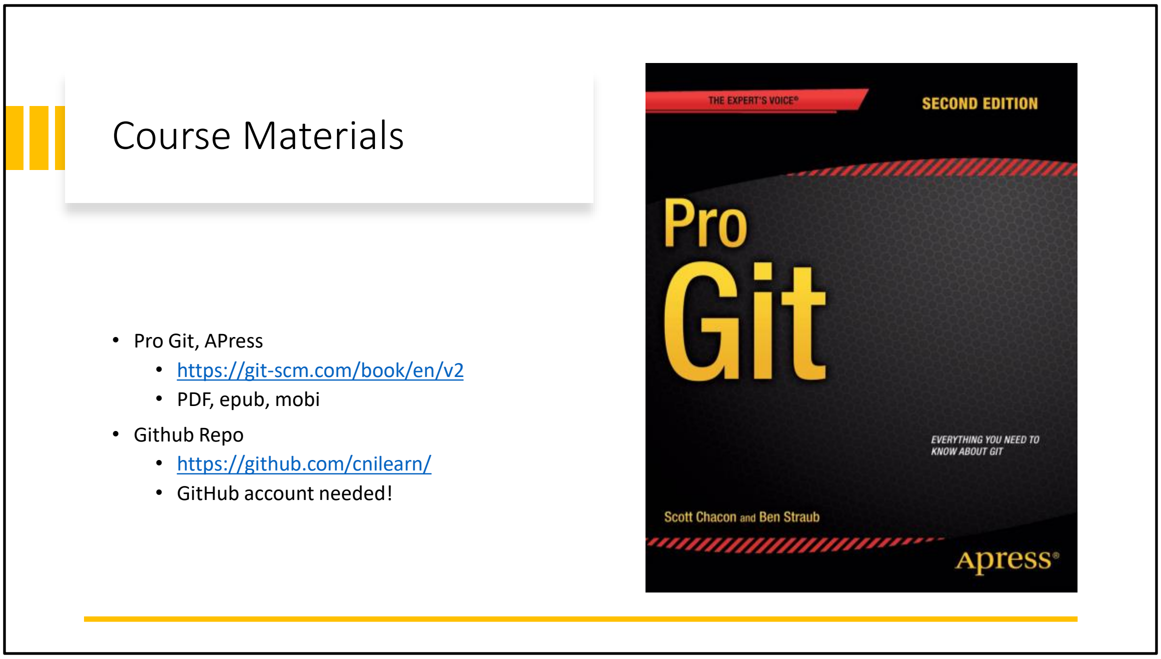

Join the discussion @ p2p.wrox.com
wrox
Wrox Programmer to Programmer™
Professional
C# 7 and
.NET Core 2.0
Christian Nagel



About you

EXPERIENCE

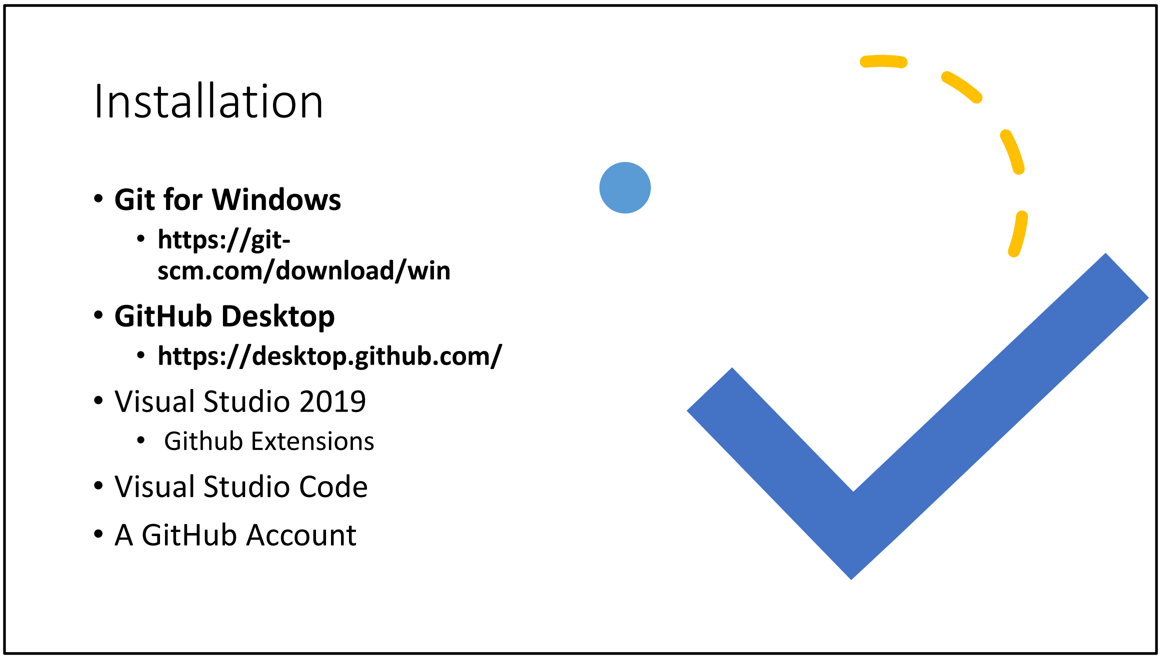
EXPECTATIONS



Course Materials

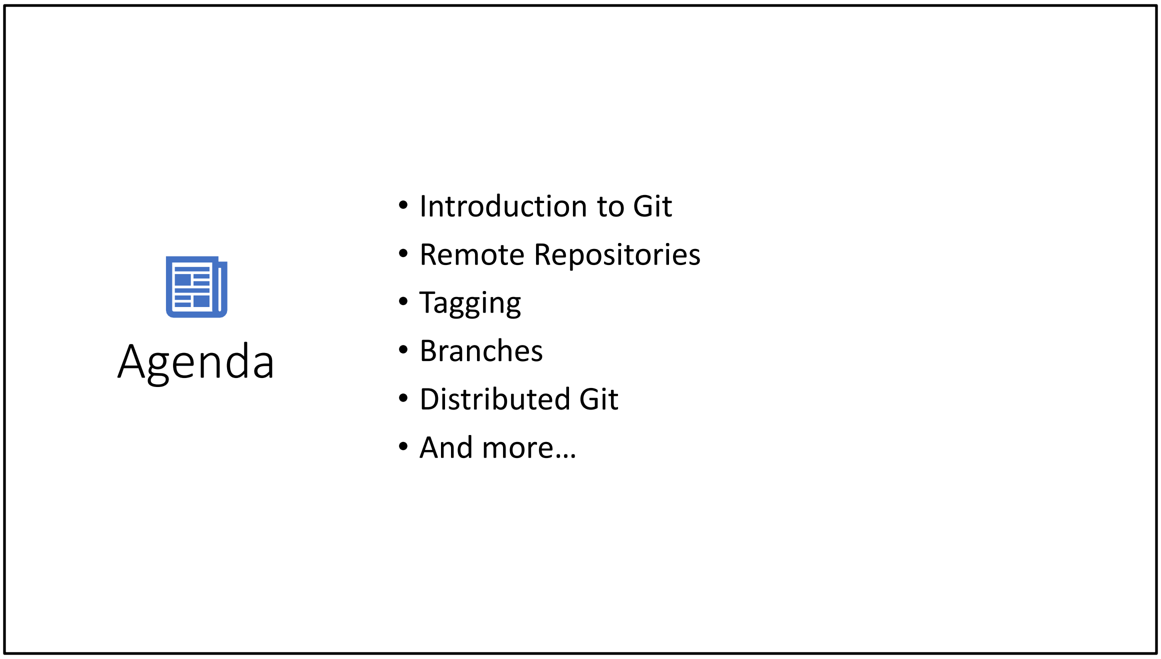
- Pro Git, APress
  - https://git-scm.com/book/en/v2
  - PDF, epub, mobi
- Github Repo
  - https://github.com/cnilearn/
  - GitHub account needed!

THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®



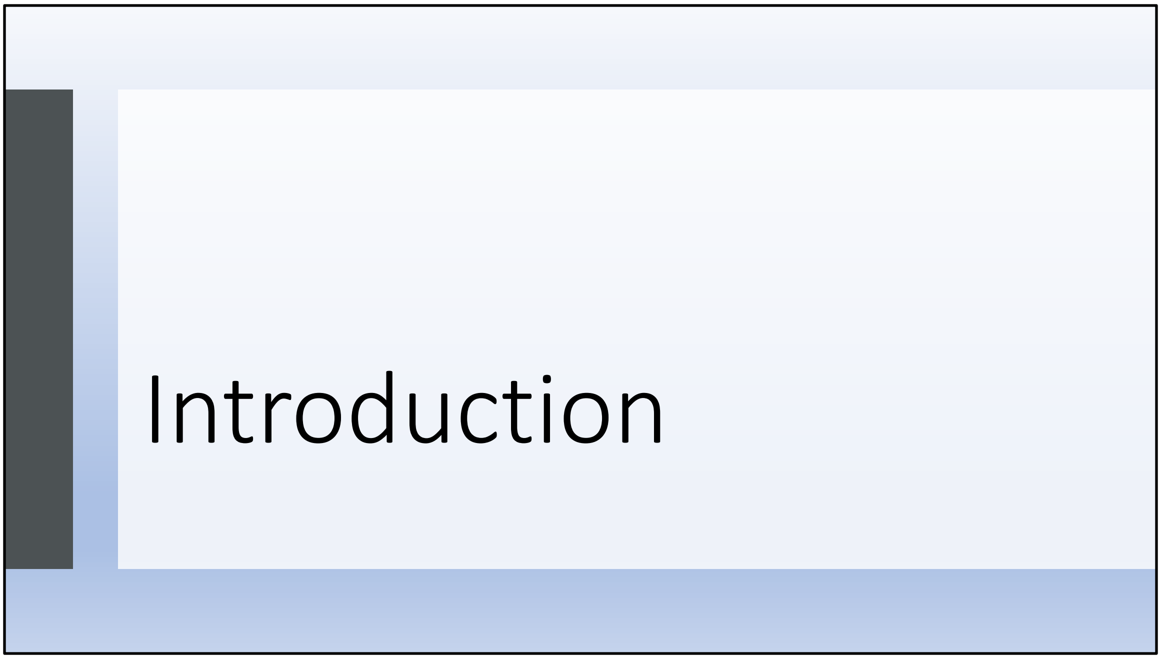
Installation

- **Git for Windows**
  - **https://git-scm.com/download/win**
- **GitHub Desktop**
  - **https://desktop.github.com/**
- Visual Studio 2019
  - Github Extensions
- Visual Studio Code
- A GitHub Account

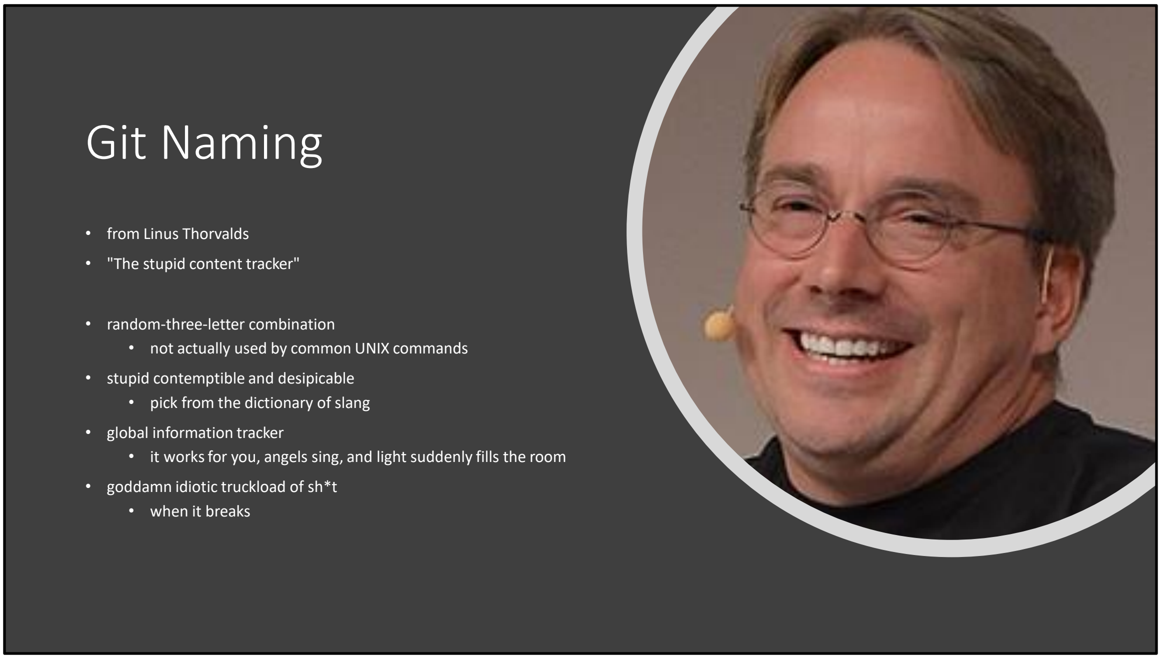


Agenda

- Introduction to Git
- Remote Repositories
- Tagging
- Branches
- Distributed Git
- And more…



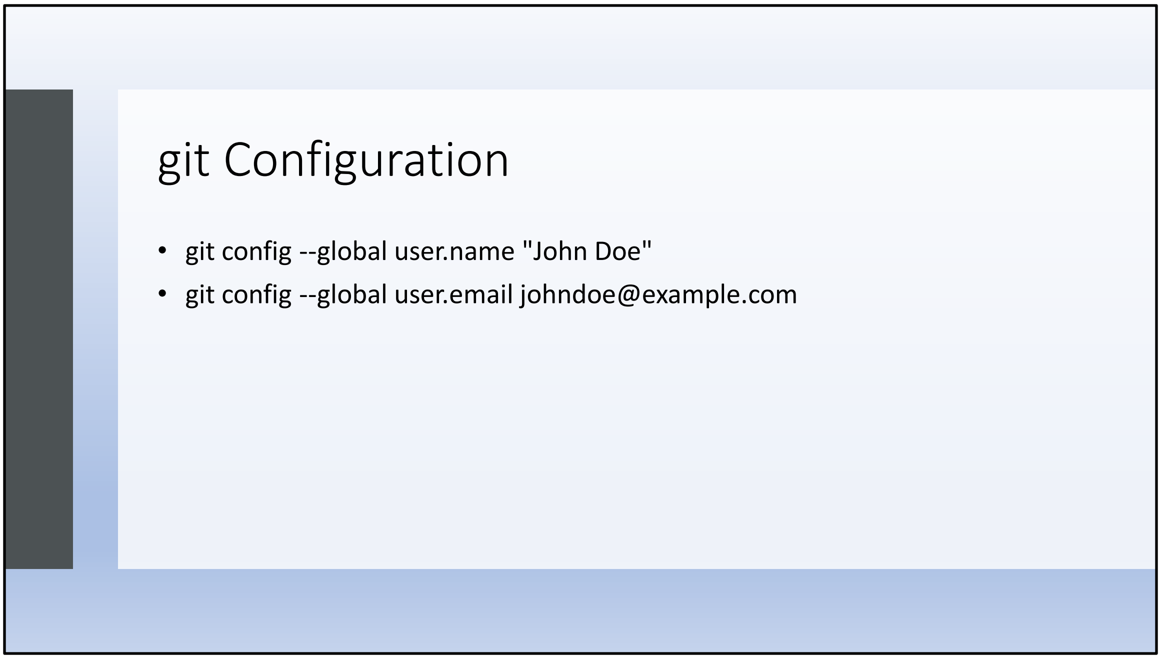
Introduction



Git Naming

- from Linus Thorvalds
- "The stupid content tracker"

- random-three-letter combination
  - not actually used by common UNIX commands
- stupid contemptible and desipicable
  - pick from the dictionary of slang
- global information tracker
  - it works for you, angels sing, and light suddenly fills the room
- goddamn idiotic truckload of sh*t
  - when it breaks

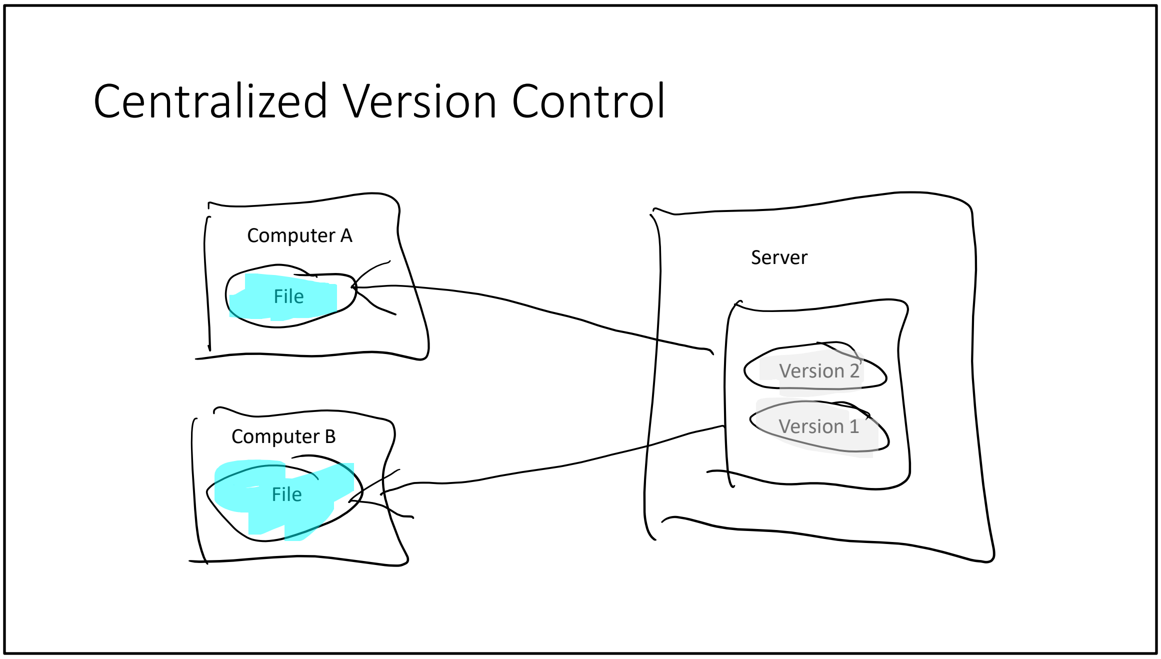


git Configuration

- git config --global user.name "John Doe"
- git config --global user.email johndoe@example.com

Help

- git --help
- git <command> --help



Centralized Version Control

Computer A

File

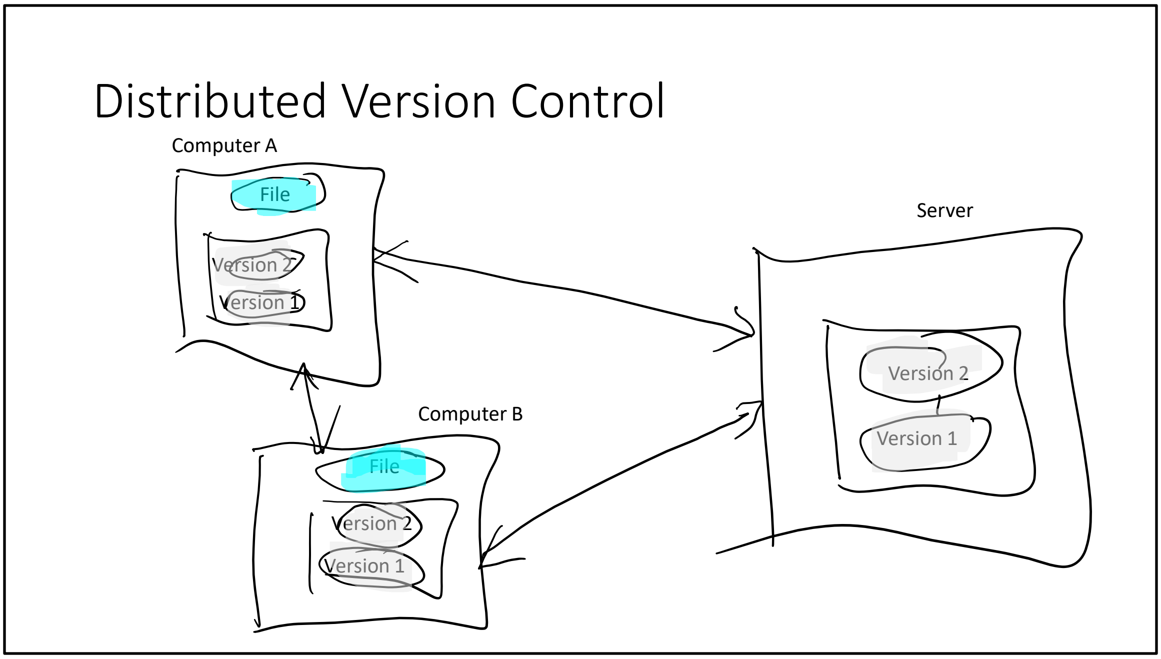
Computer B

File

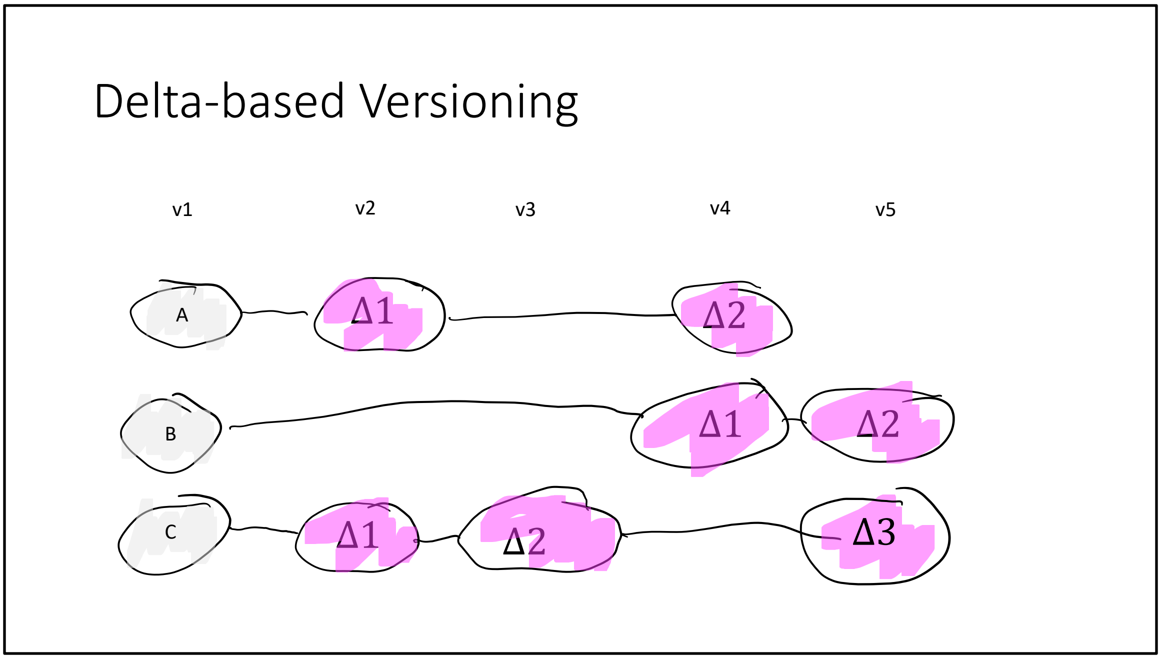
Server

Version 2

Version 1

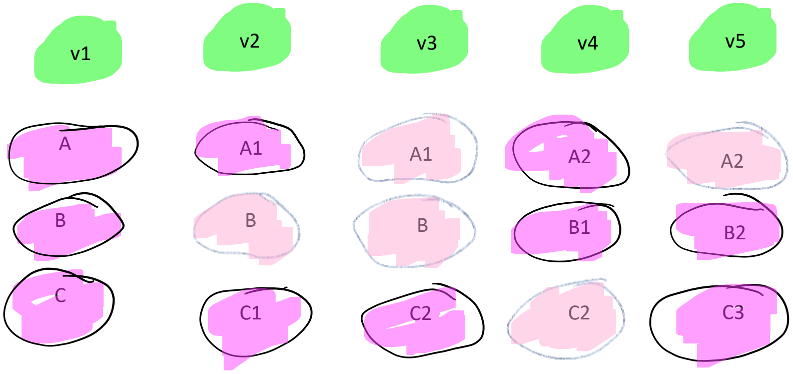


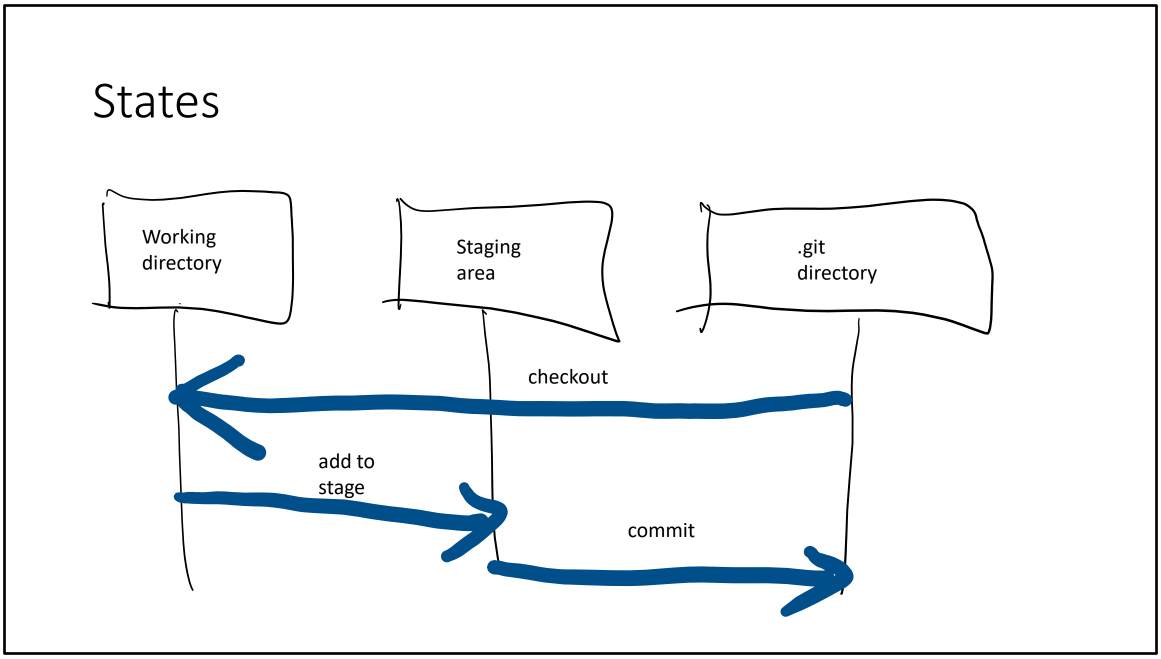
Distributed Version Control



Delta-based Versioning

Snapshots





States

Working directory

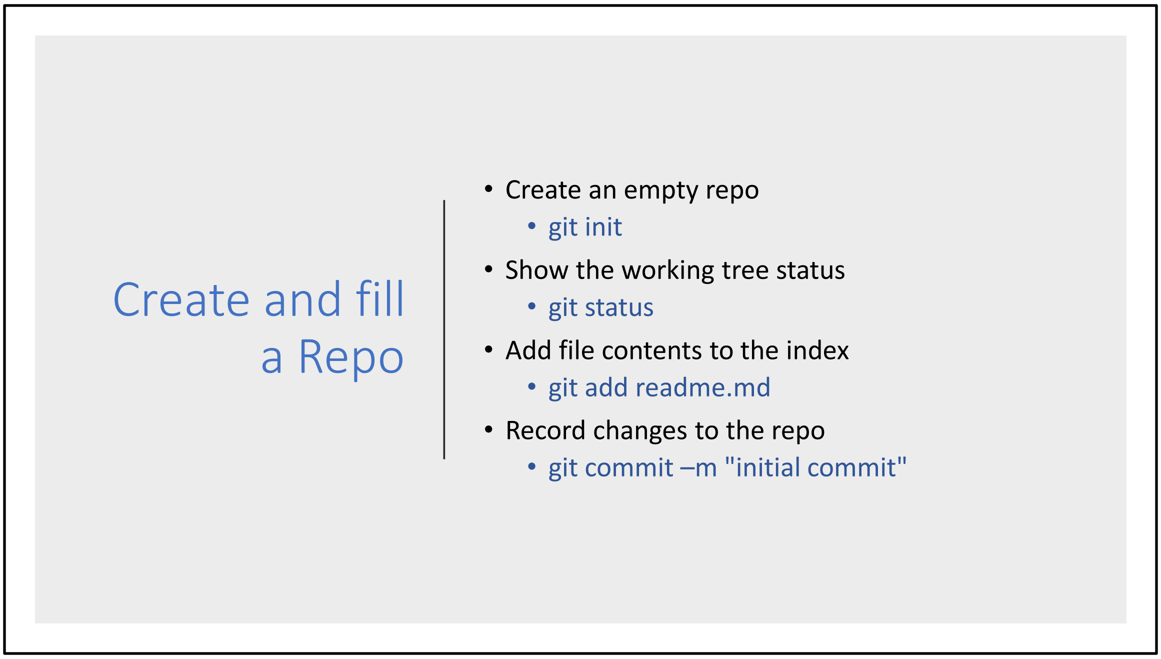
Staging area

.git directory

checkout

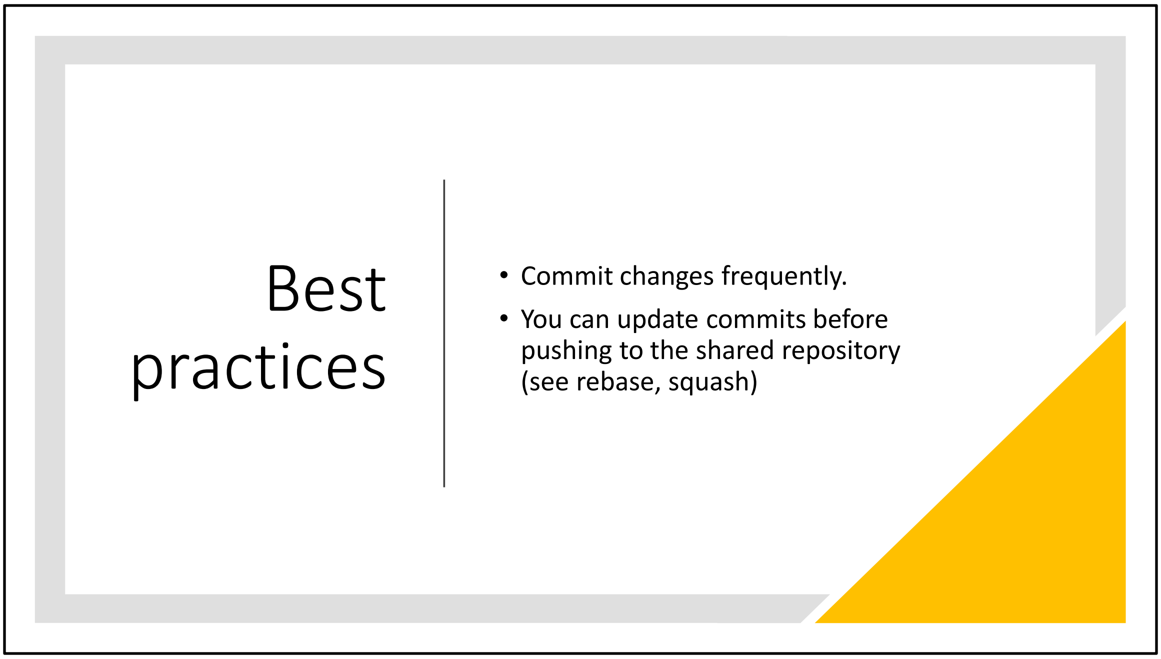
add to stage

commit



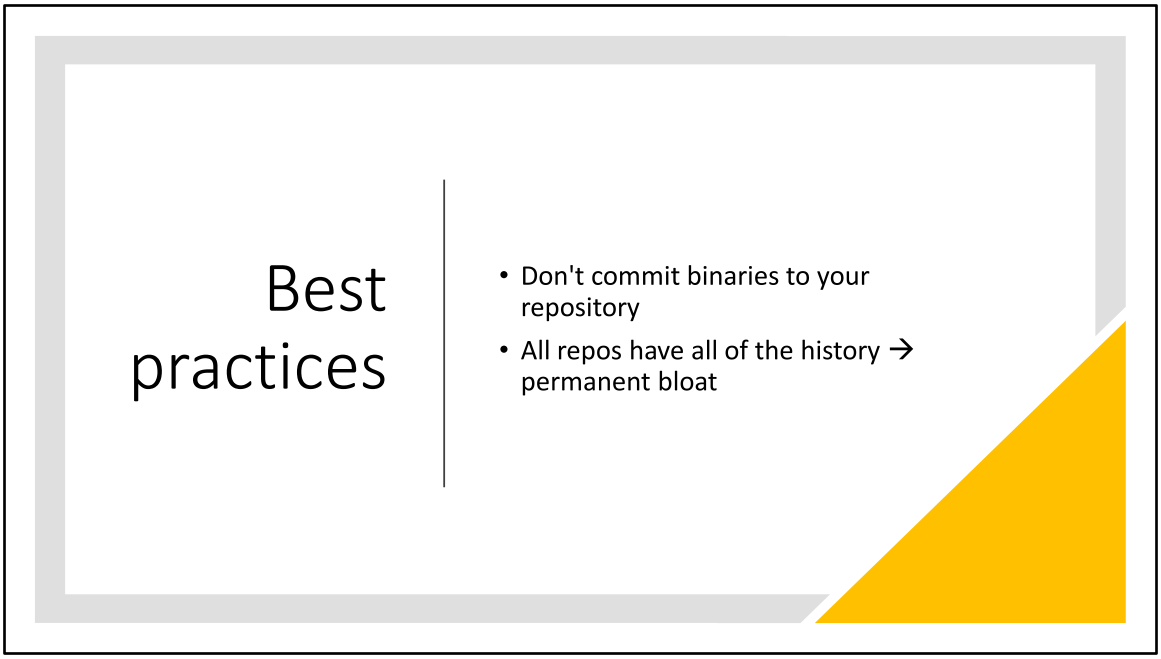
Create and fill a Repo

- Create an empty repo
  - git init
- Show the working tree status
  - git status
- Add file contents to the index
  - git add readme.md
- Record changes to the repo
  - git commit –m "initial commit"



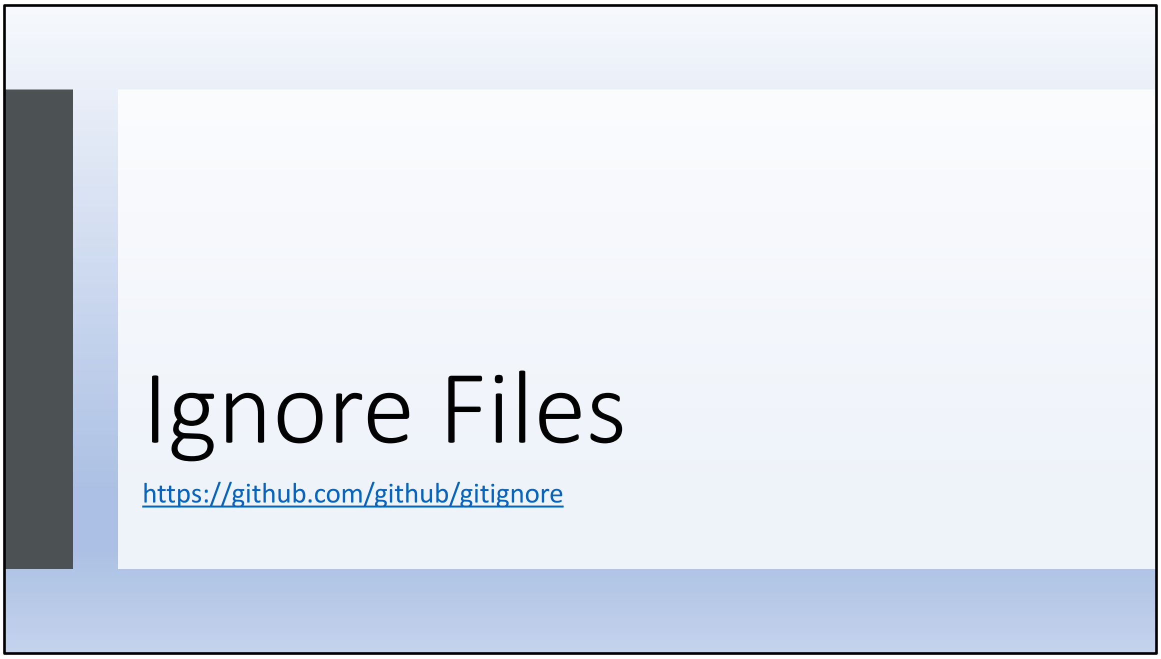
Best practices

- Commit changes frequently.
- You can update commits before pushing to the shared repository (see rebase, squash)



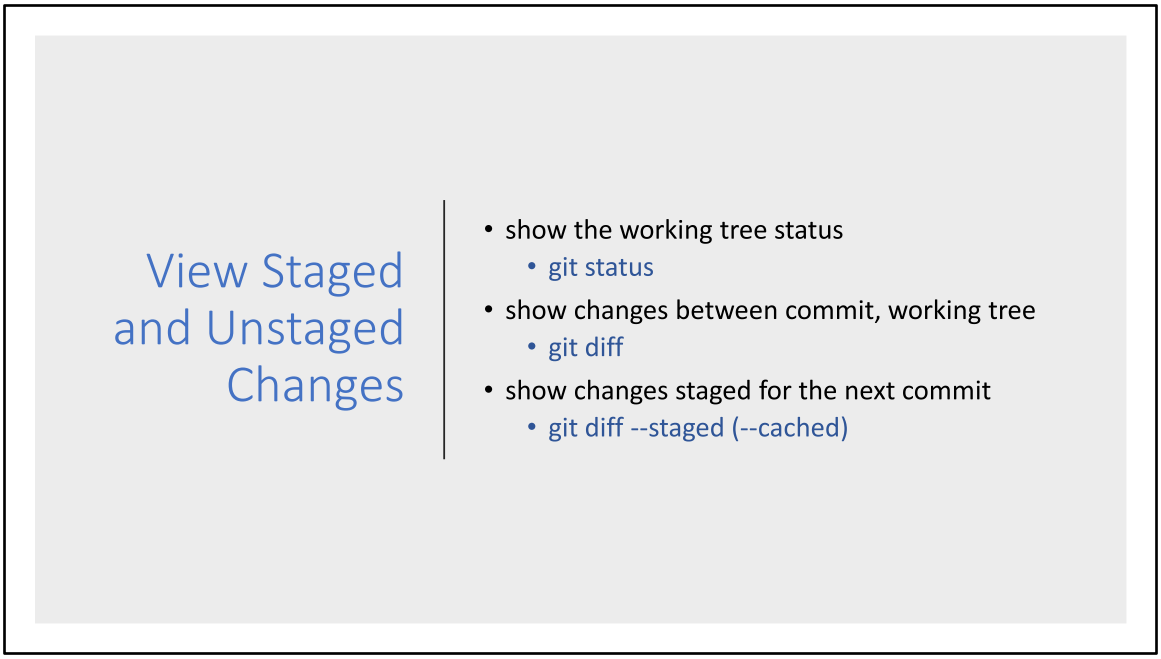
Best practices

- Don't commit binaries to your repository
- All repos have all of the history → permanent bloat



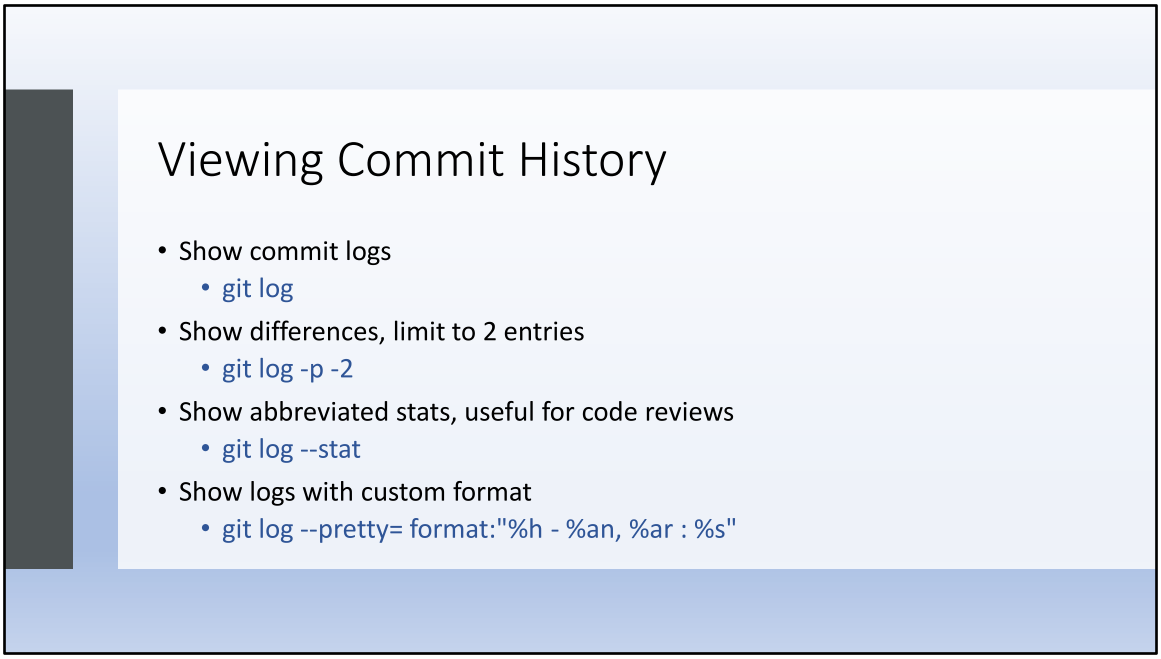
Ignore Files

https://github.com/github/gitignore



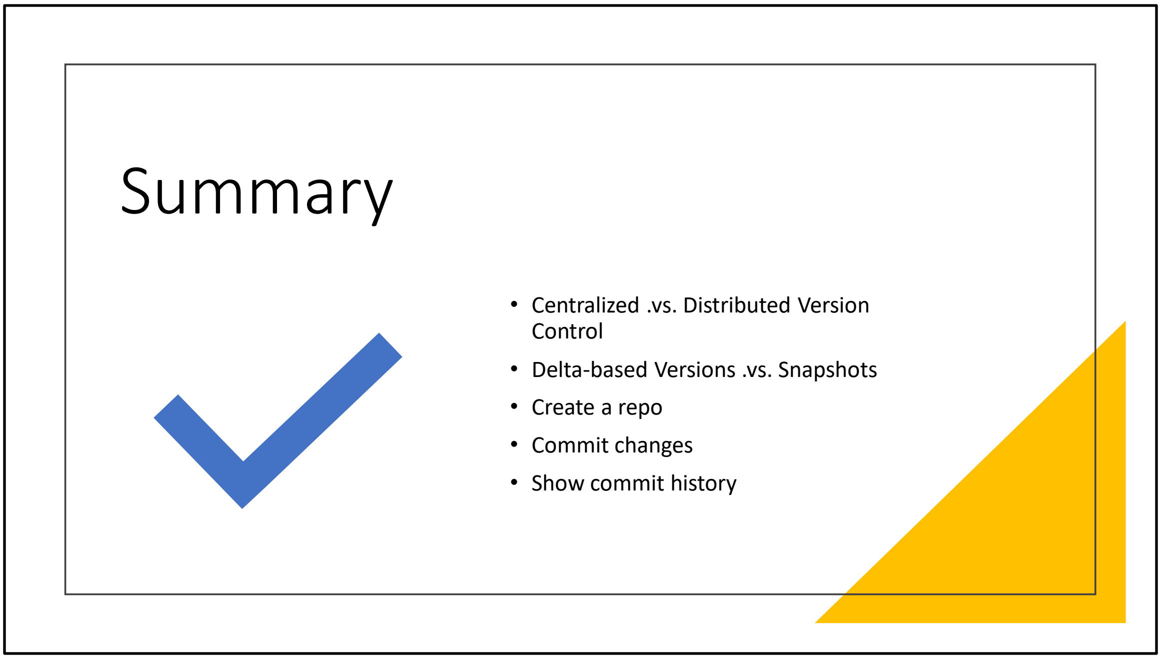
View Staged and Unstaged Changes

- show the working tree status
  - git status
- show changes between commit, working tree
  - git diff
- show changes staged for the next commit
  - git diff --staged (--cached)



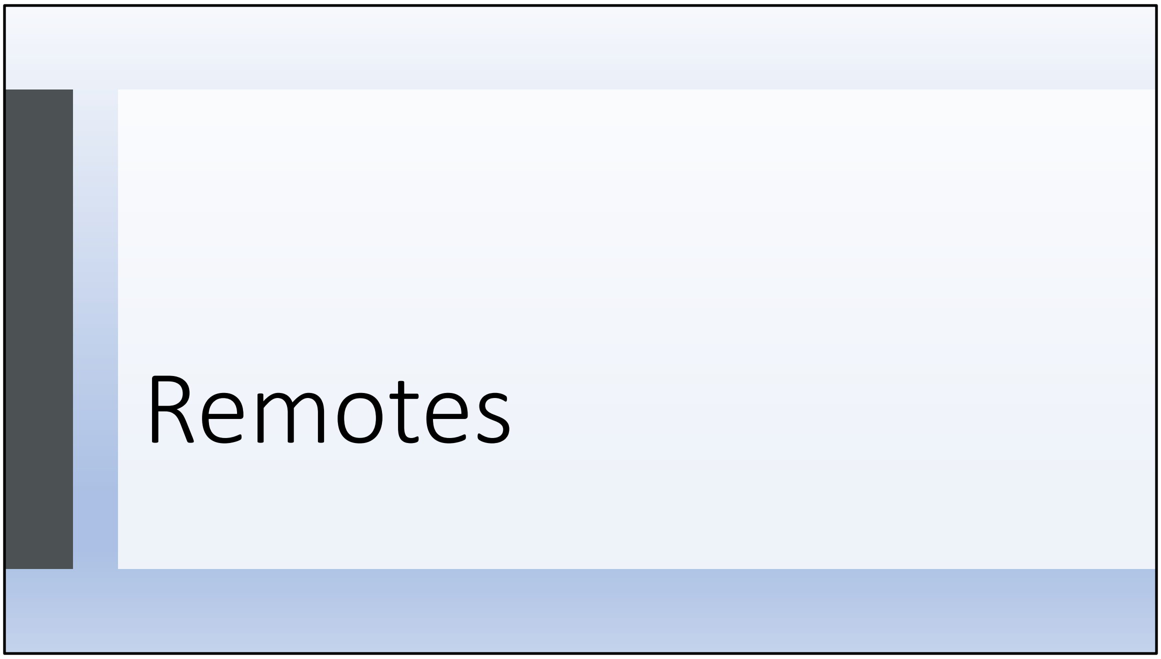
Viewing Commit History

- Show commit logs
  - git log
- Show differences, limit to 2 entries
  - git log -p -2
- Show abbreviated stats, useful for code reviews
  - git log --stat
- Show logs with custom format
  - git log --pretty= format:"%h - %an, %ar : %s"

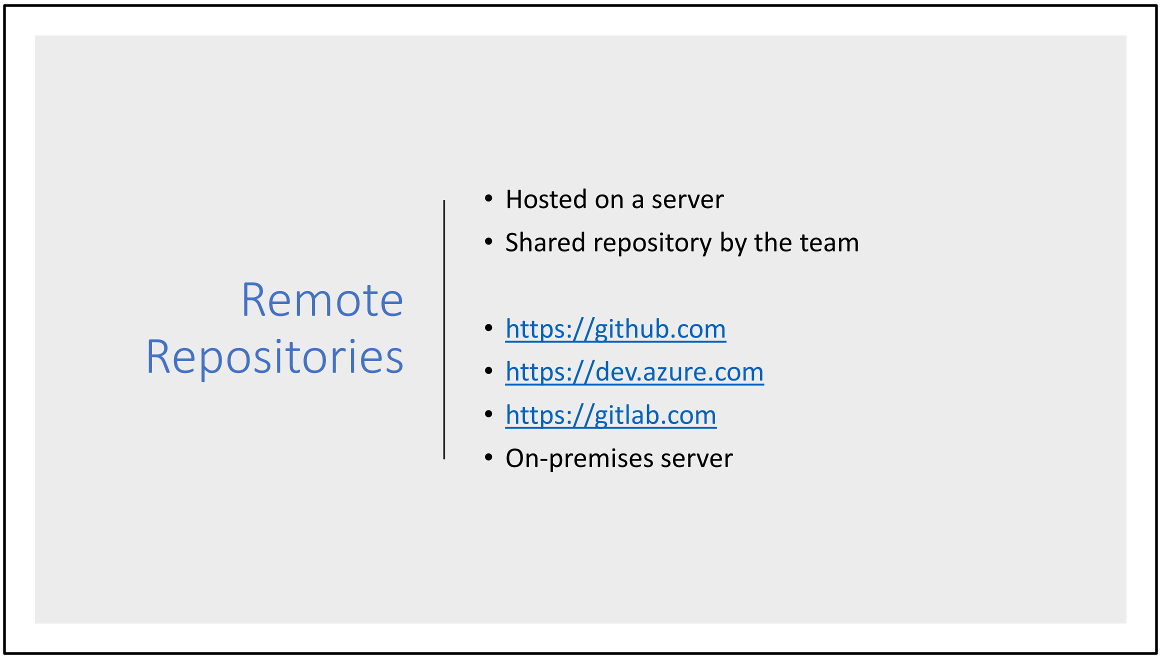


Summary

- Centralized .vs. Distributed Version Control
- Delta-based Versions .vs. Snapshots
- Create a repo
- Commit changes
- Show commit history



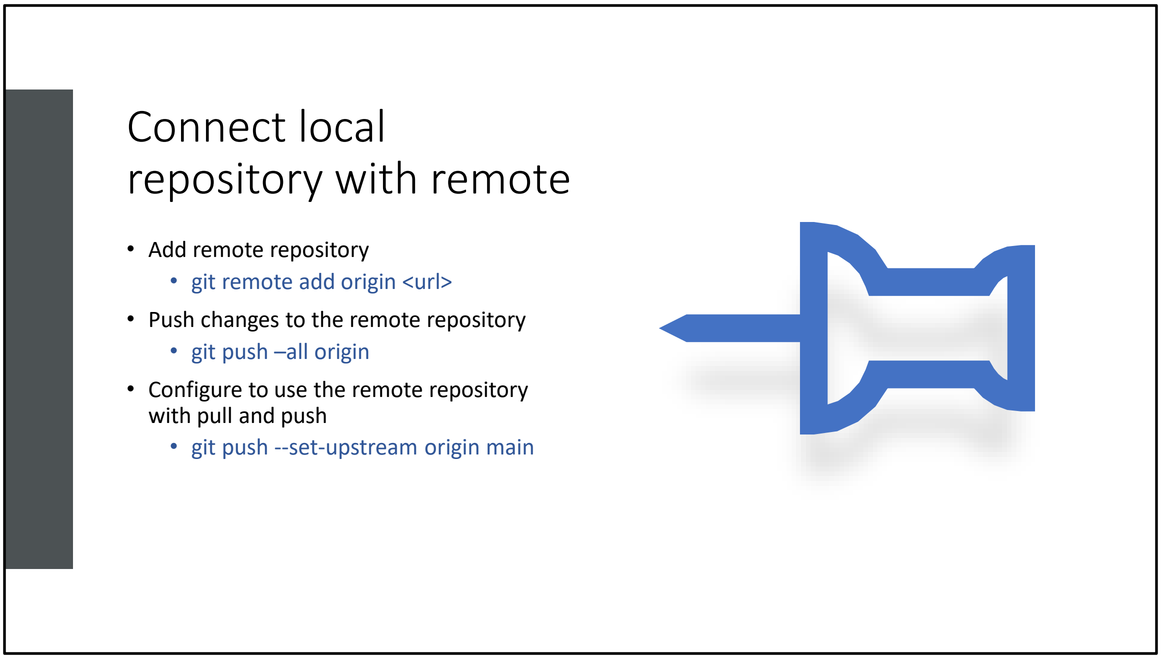
Remotes



Remote Repositories

- Hosted on a server
- Shared repository by the team

- https://github.com
- https://dev.azure.com
- https://gitlab.com
- On-premises server

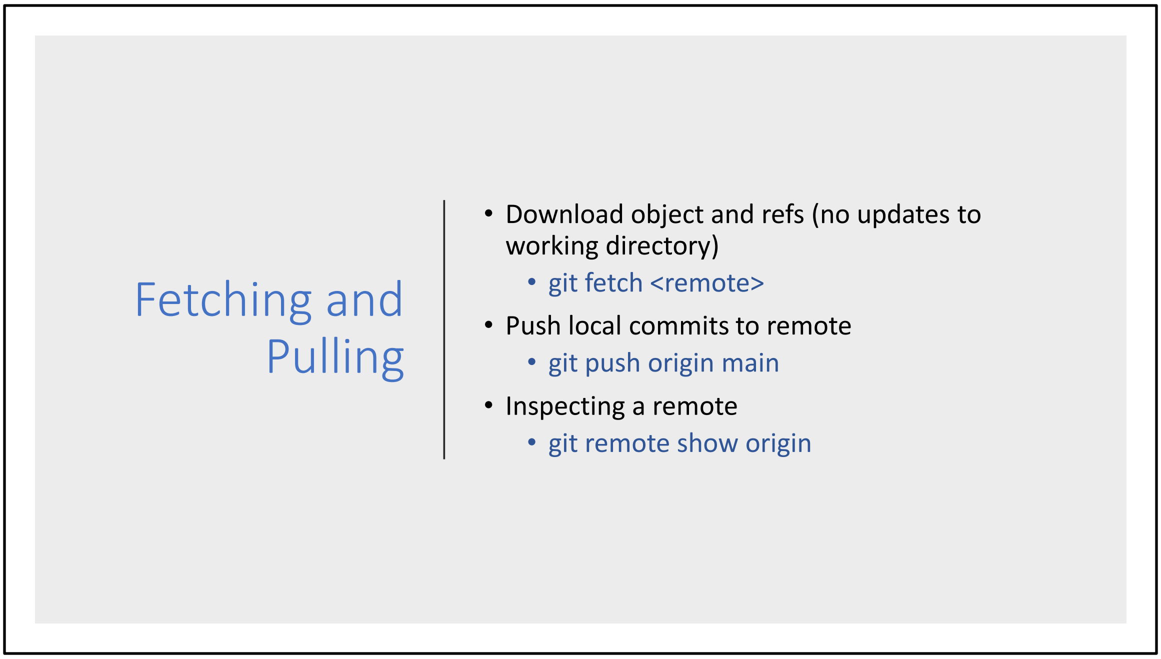


Connect local repository with remote

- Add remote repository
  - git remote add origin <url>
- Push changes to the remote repository
  - git push –all origin
- Configure to use the remote repository with pull and push
  - git push --set-upstream origin main

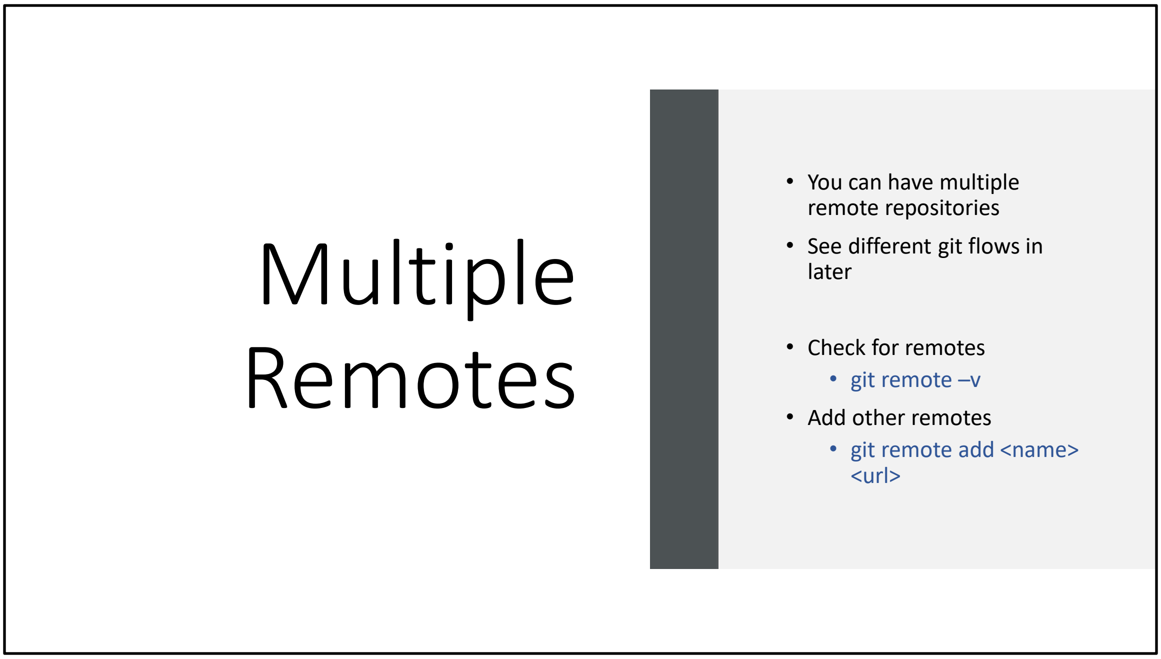
Clone a remote repo

- Clone a remote repo
  - git clone <repo>
- Show remotes
  - git remote -v



Fetching and Pulling

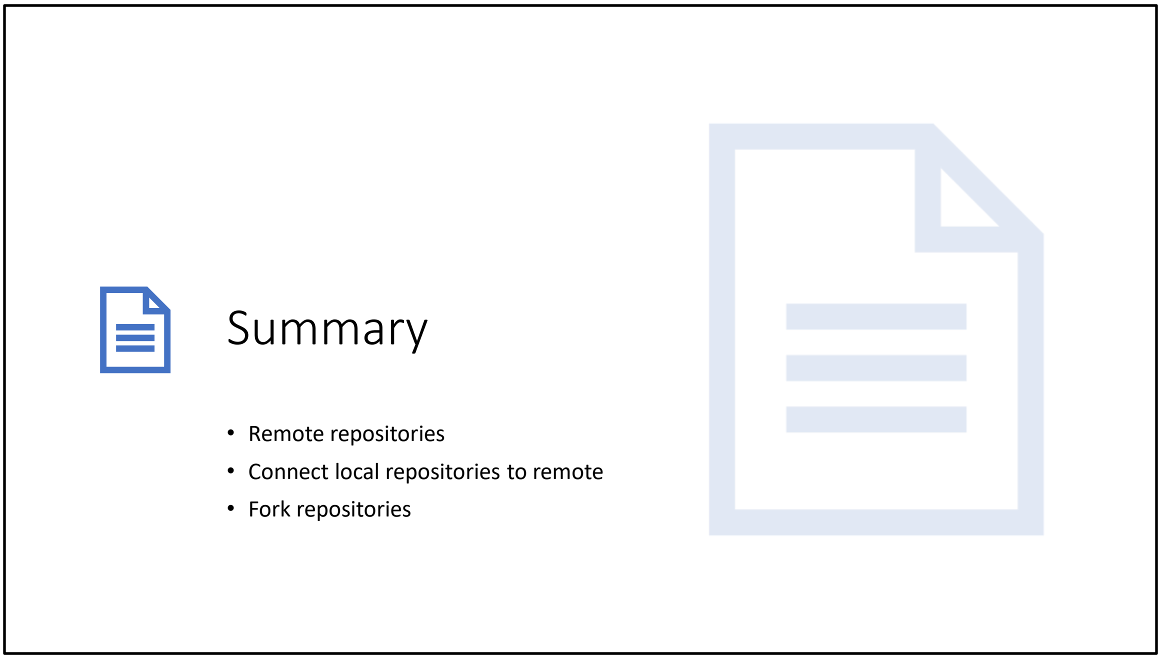
- Download object and refs (no updates to working directory)
  - git fetch <remote>
- Push local commits to remote
  - git push origin main
- Inspecting a remote
  - git remote show origin



Multiple Remotes

- You can have multiple remote repositories
- See different git flows in later

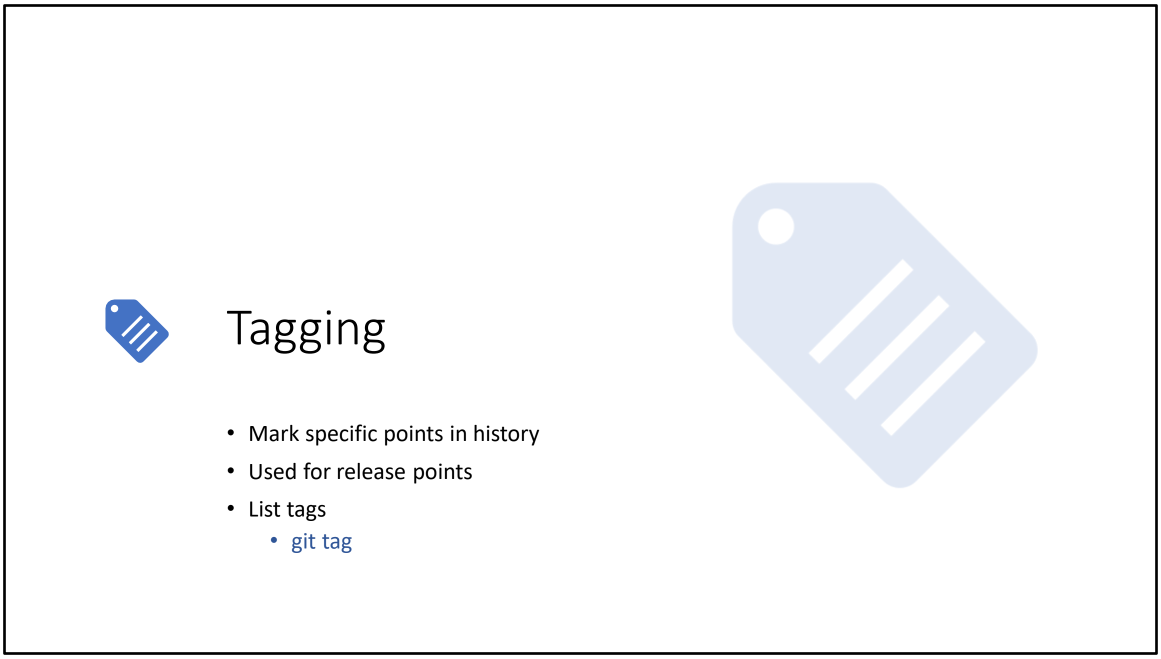
- Check for remotes
  - git remote –v
- Add other remotes
  - git remote add <name> <url>



Summary

- Remote repositories
- Connect local repositories to remote
- Fork repositories

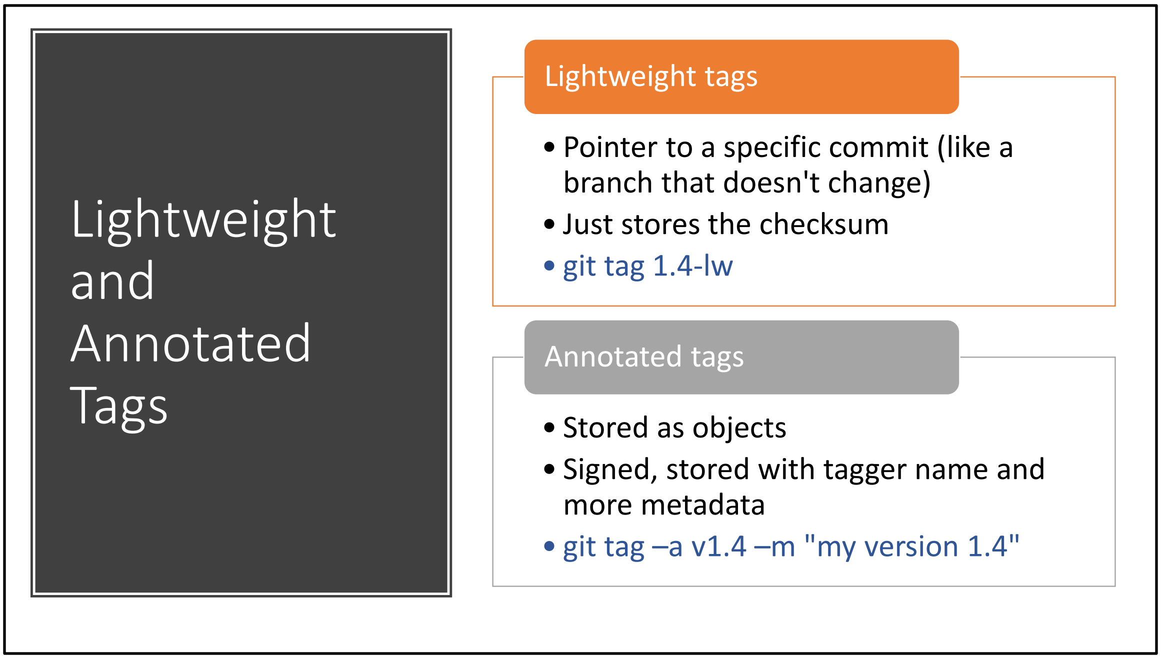
Tagging



Tagging

- Mark specific points in history
- Used for release points
- List tags
  - git tag

show tags at https://github.com/dotnet/aspnetcore/



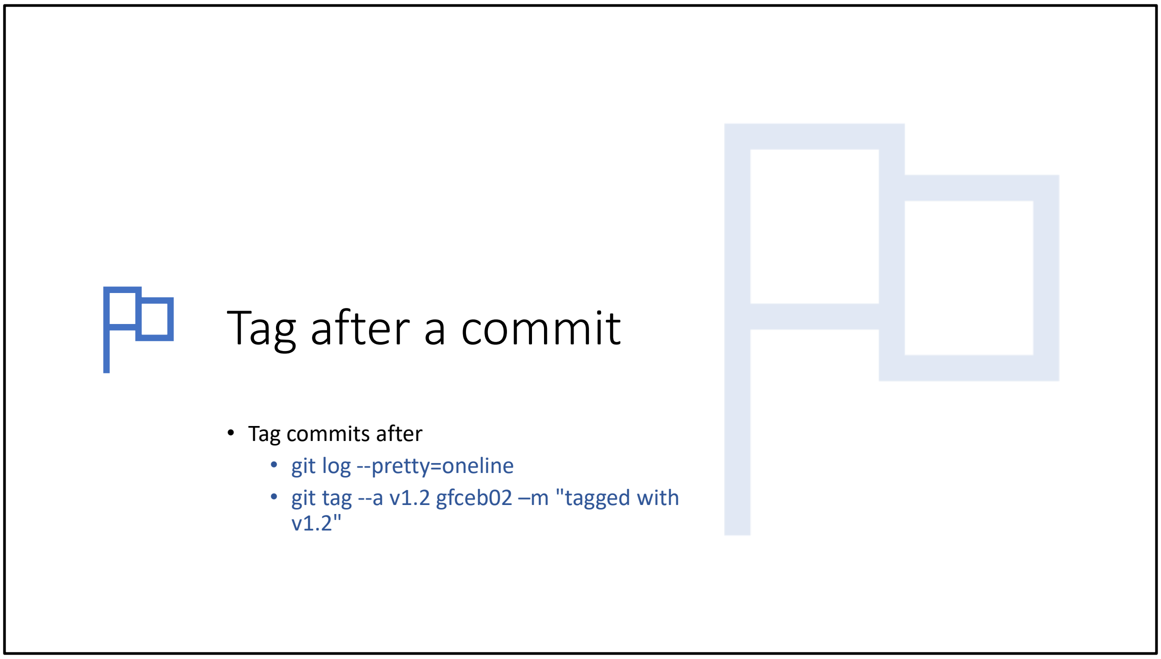
Lightweight and Annotated Tags

Lightweight tags

- Pointer to a specific commit (like a branch that doesn't change)
- Just stores the checksum
- git tag 1.4-lw

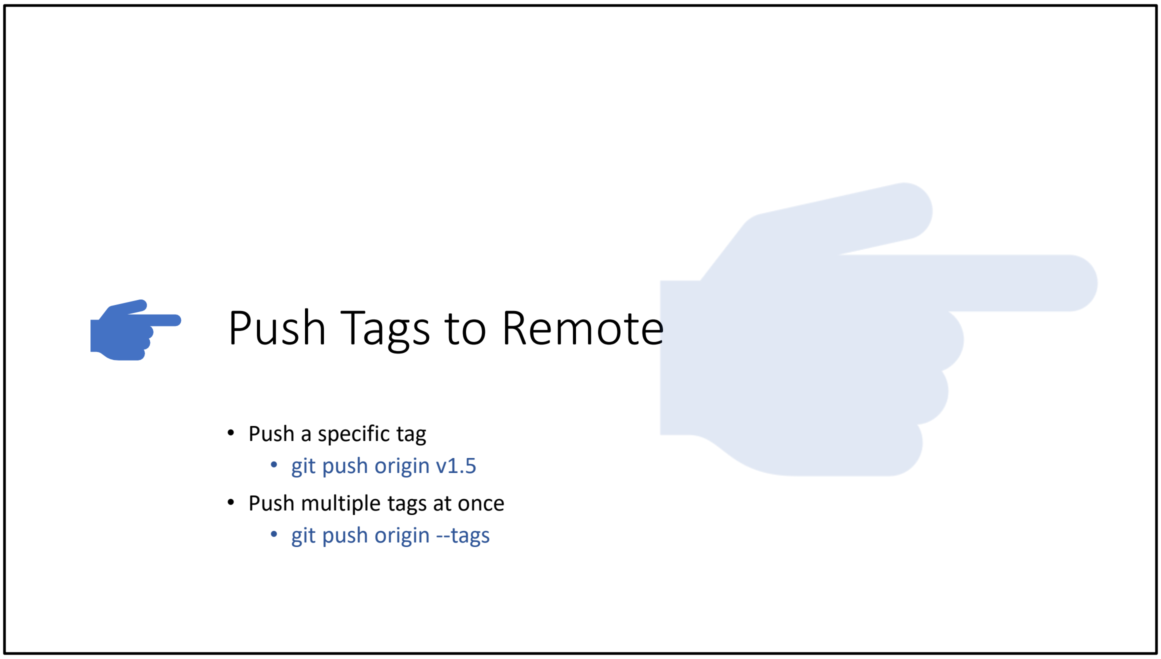
Annotated tags

- Stored as objects
- Signed, stored with tagger name and more metadata
- git tag –a v1.4 –m "my version 1.4"



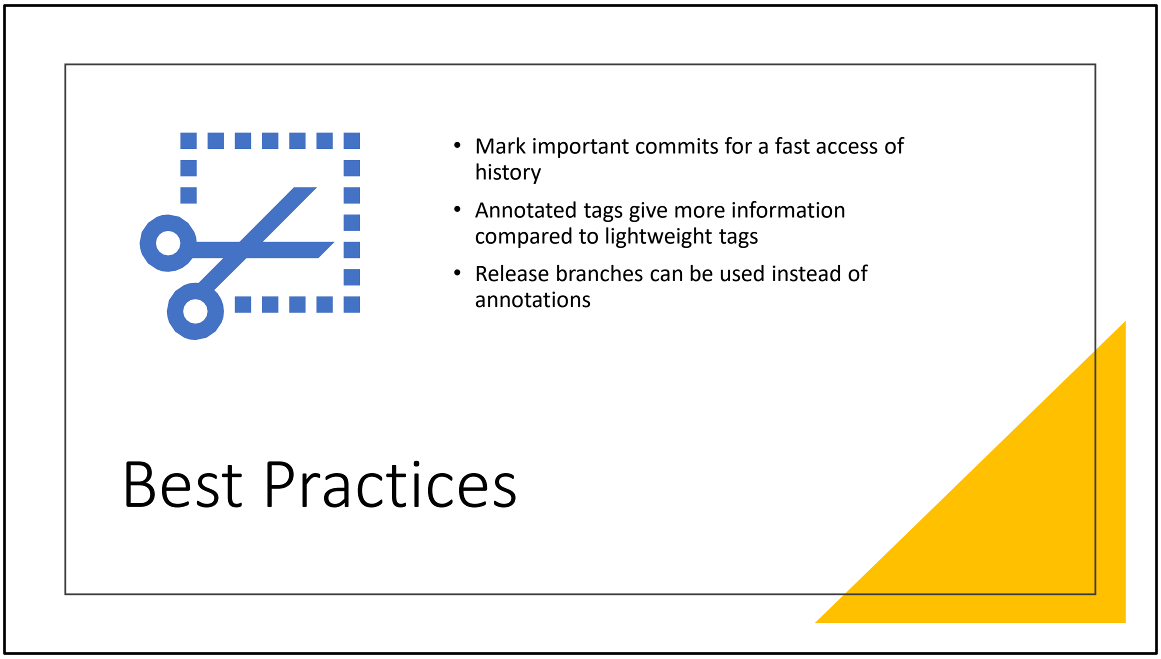
Tag after a commit

- Tag commits after
  - git log --pretty=oneline
  - git tag --a v1.2 gfceb02 –m "tagged with v1.2"



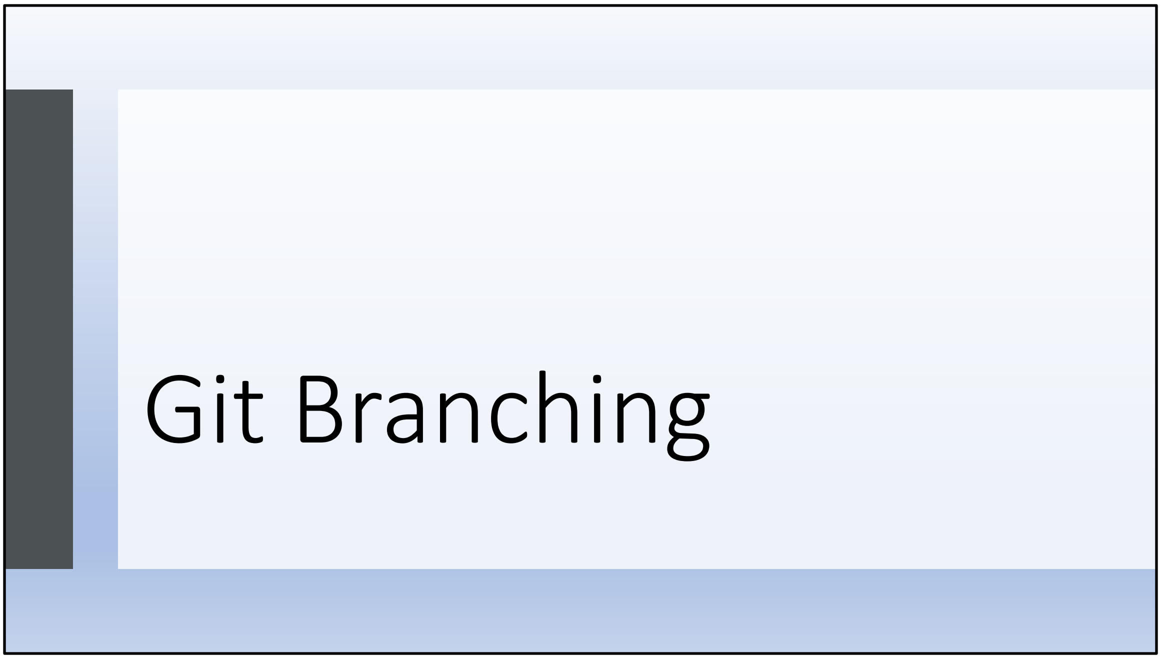
Push Tags to Remote

- Push a specific tag
  - git push origin v1.5
- Push multiple tags at once
  - git push origin --tags

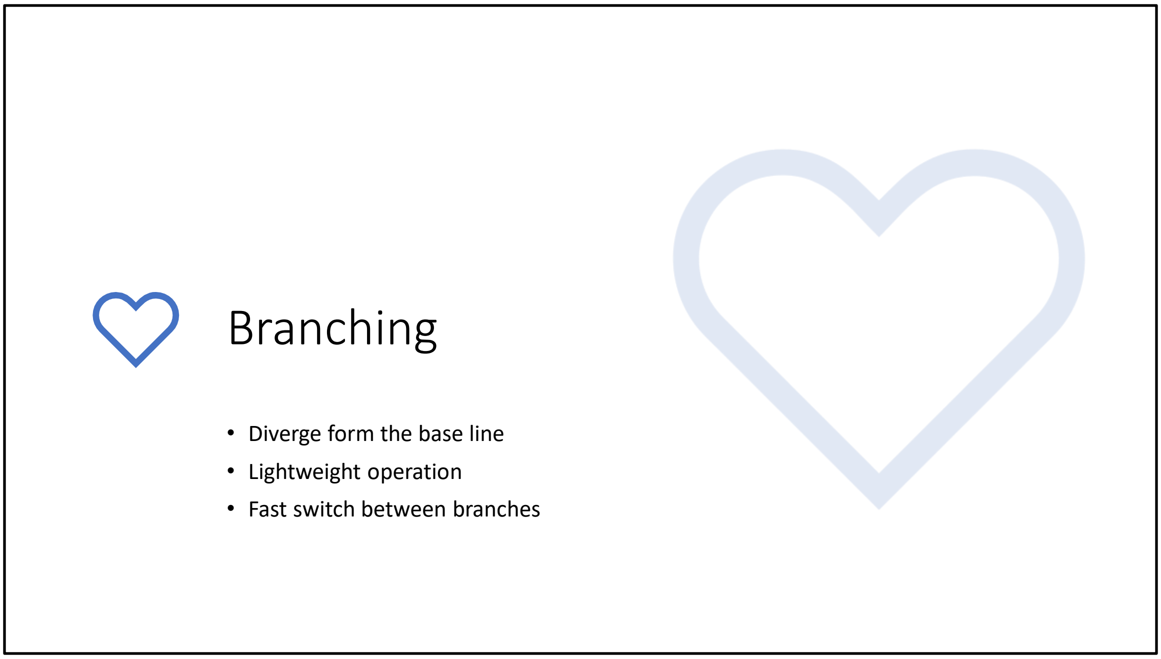


Best Practices

- Mark important commits for a fast access of history
- Annotated tags give more information compared to lightweight tags
- Release branches can be used instead of annotations

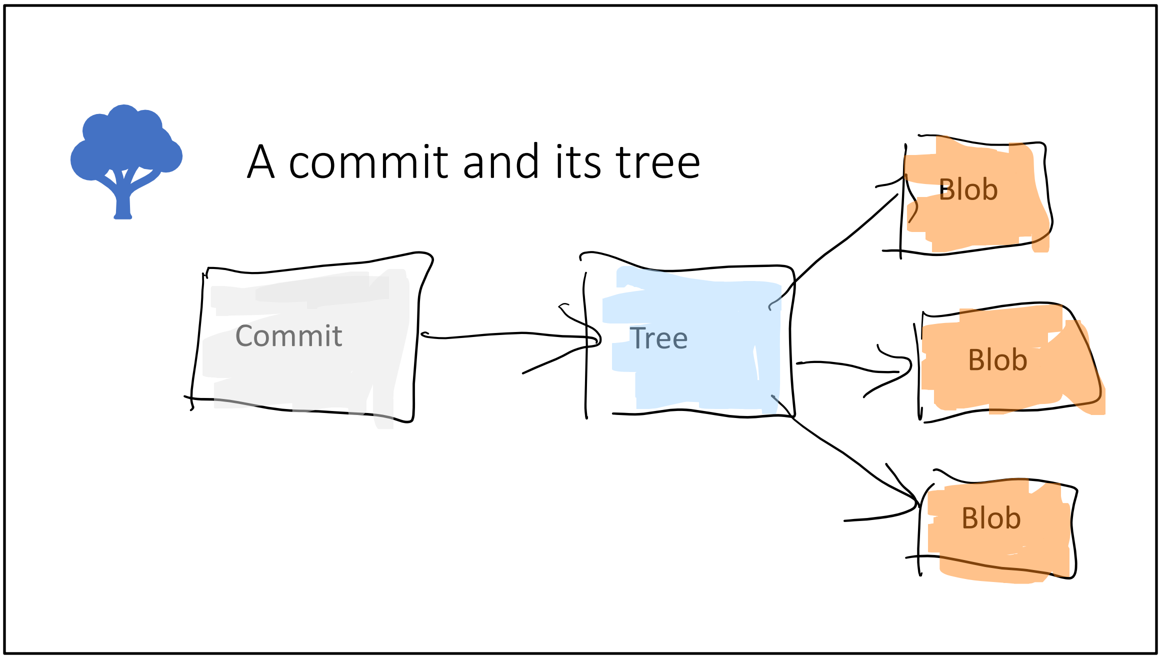


Git Branching



Branching

- Diverge form the base line
- Lightweight operation
- Fast switch between branches



A commit and its tree

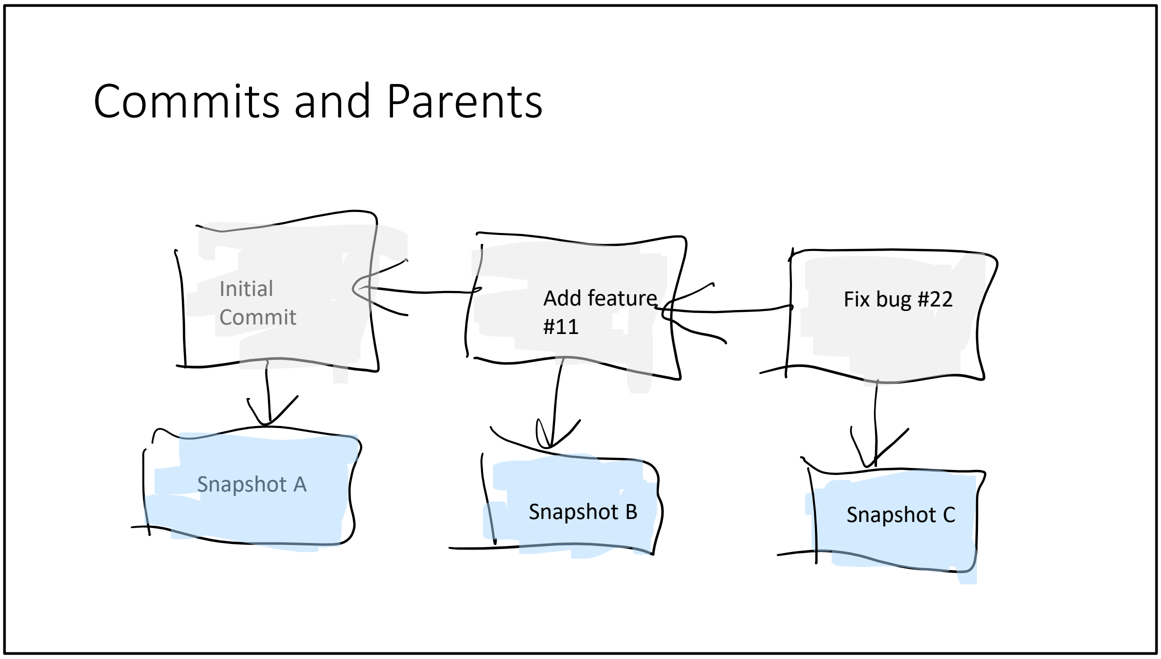
Commit

Tree

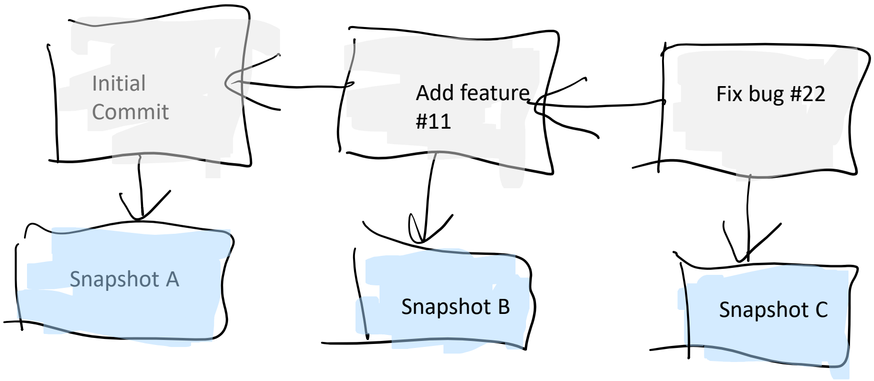
Blob

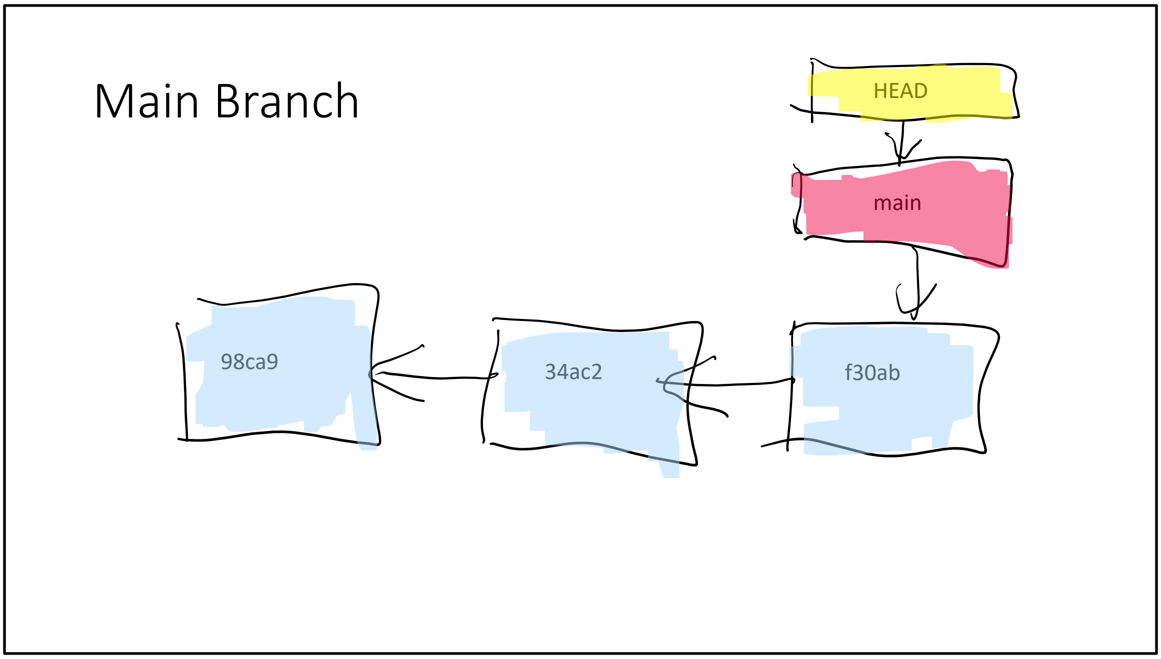
Blob

Blob



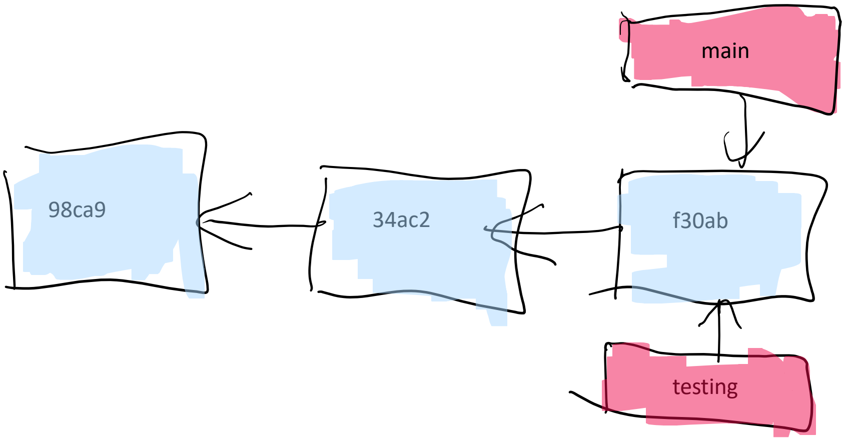
Commits and Parents

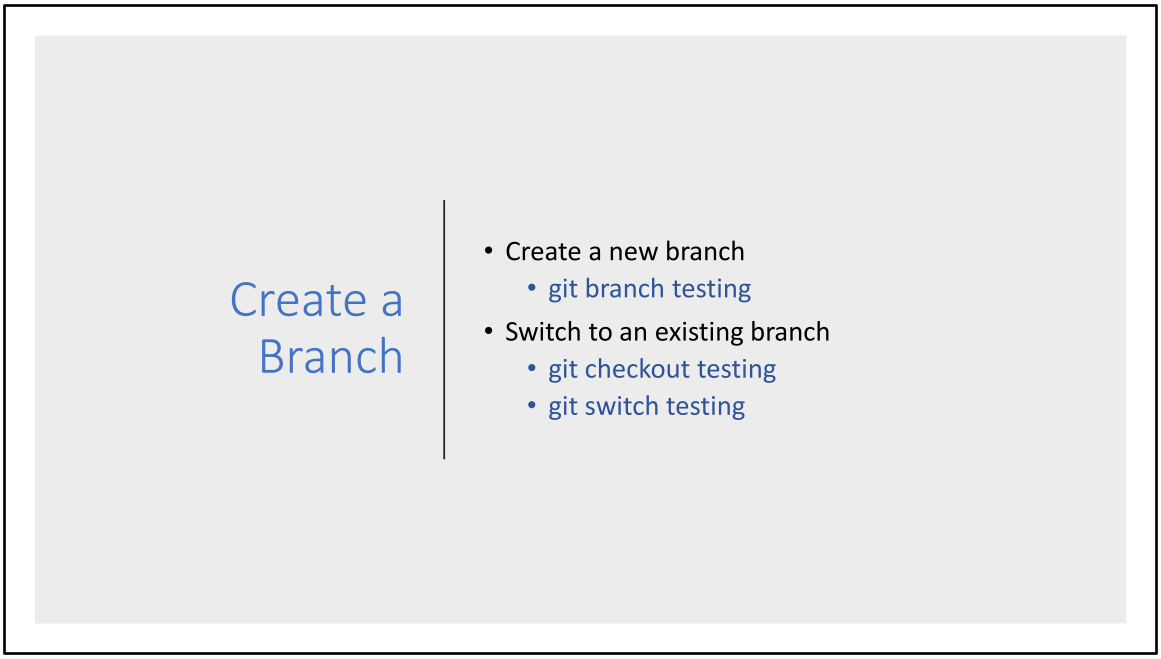




Main Branch

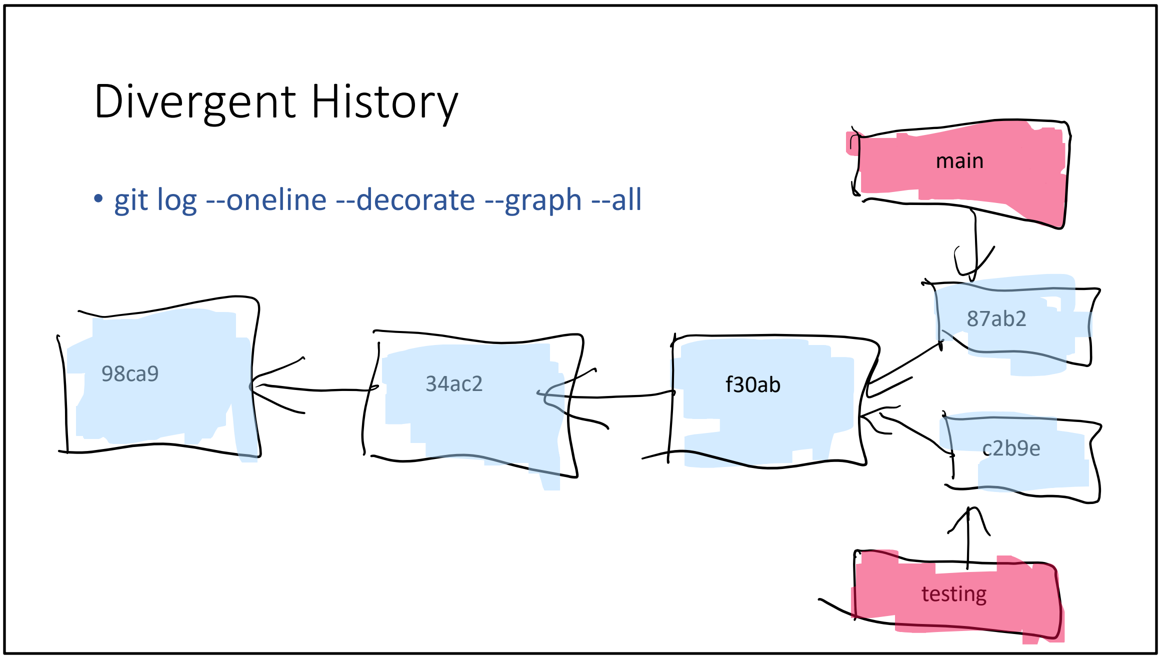
Creating a New Branch





Create a Branch

- Create a new branch
  - git branch testing
- Switch to an existing branch
  - git checkout testing
  - git switch testing



Divergent History

- git log --oneline --decorate --graph --all

main

98ca9

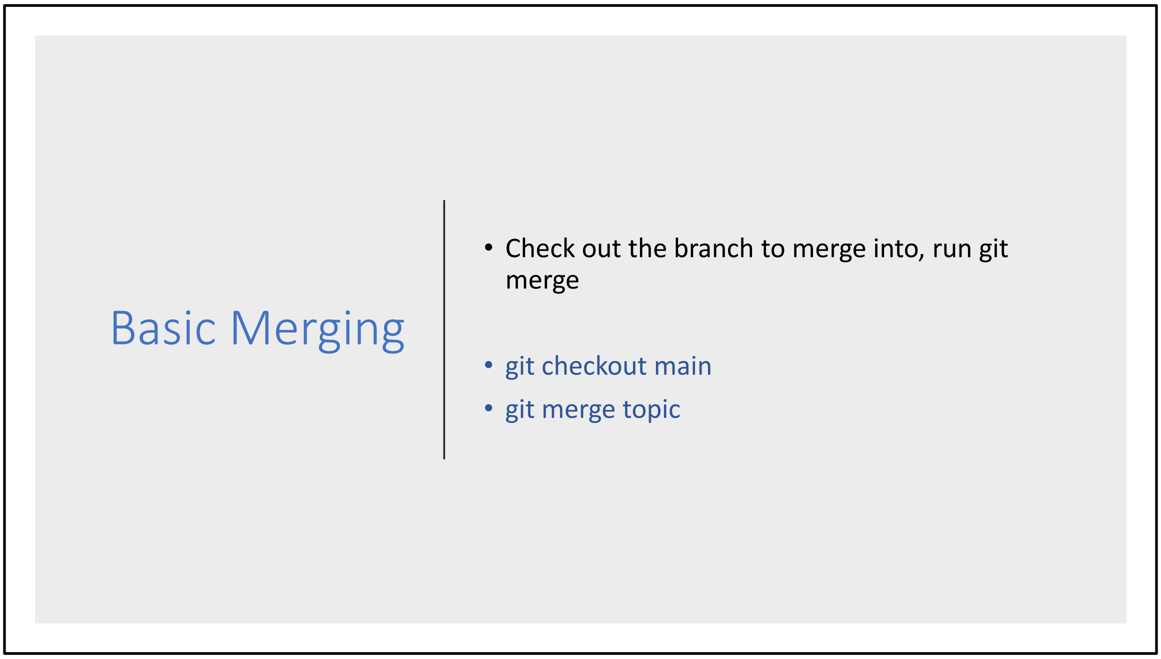
34ac2

f30ab

87ab2

c2b9e

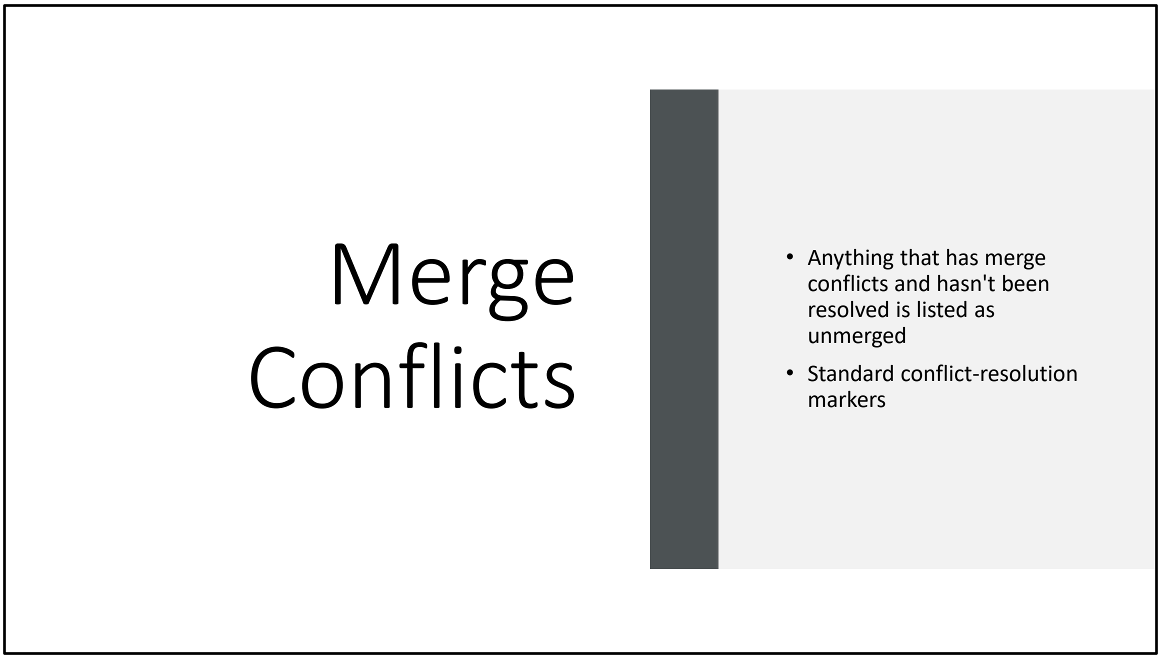
testing



Basic Merging

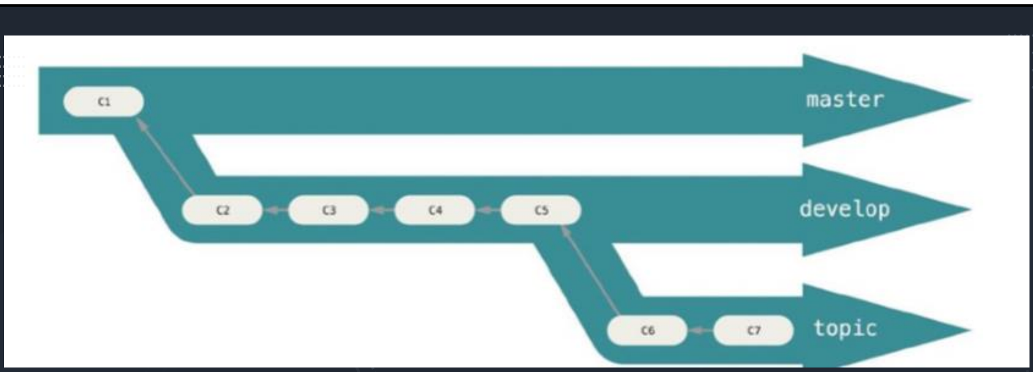
- Check out the branch to merge into, run git merge

- git checkout main
- git merge topic



Merge Conflicts

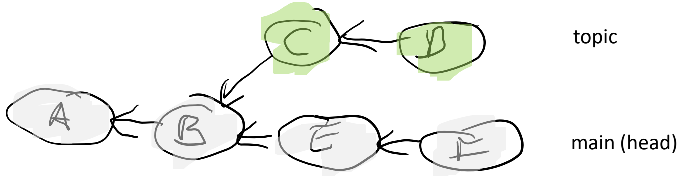
- Anything that has merge conflicts and hasn't been resolved is listed as unmerged
- Standard conflict-resolution markers



Long Running Branches

- Several branches that are always open can be used
- Sets of commits graduate

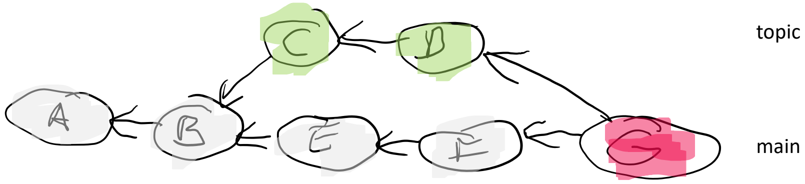
Merging Topic Branches



topic

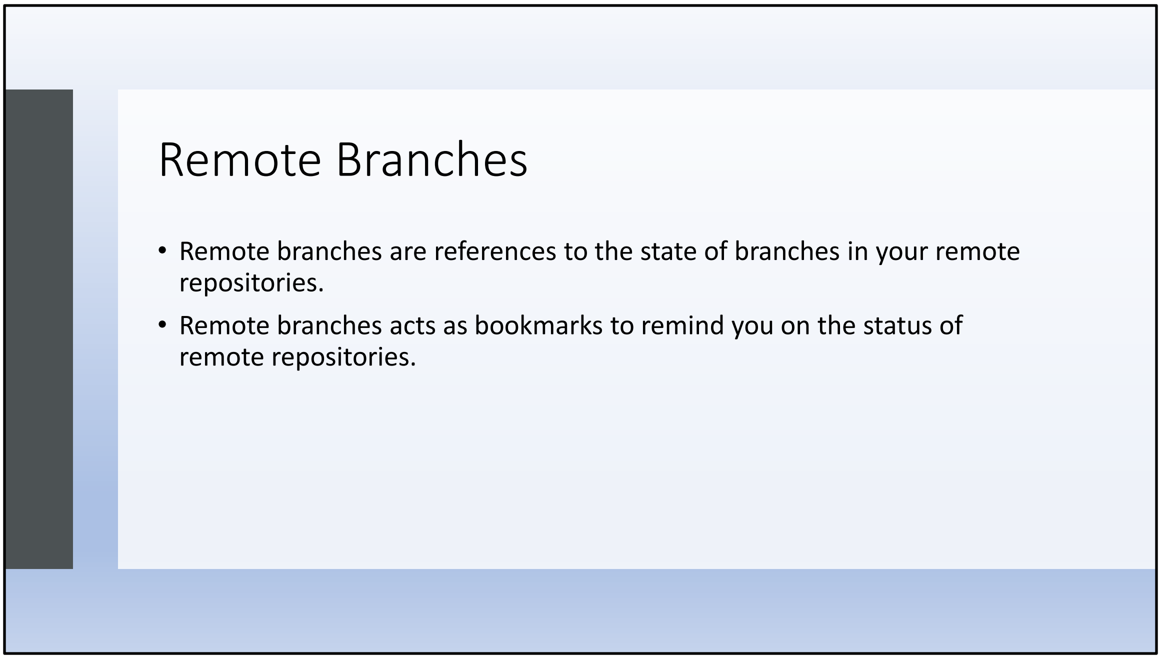
main (head)

- git merge topic



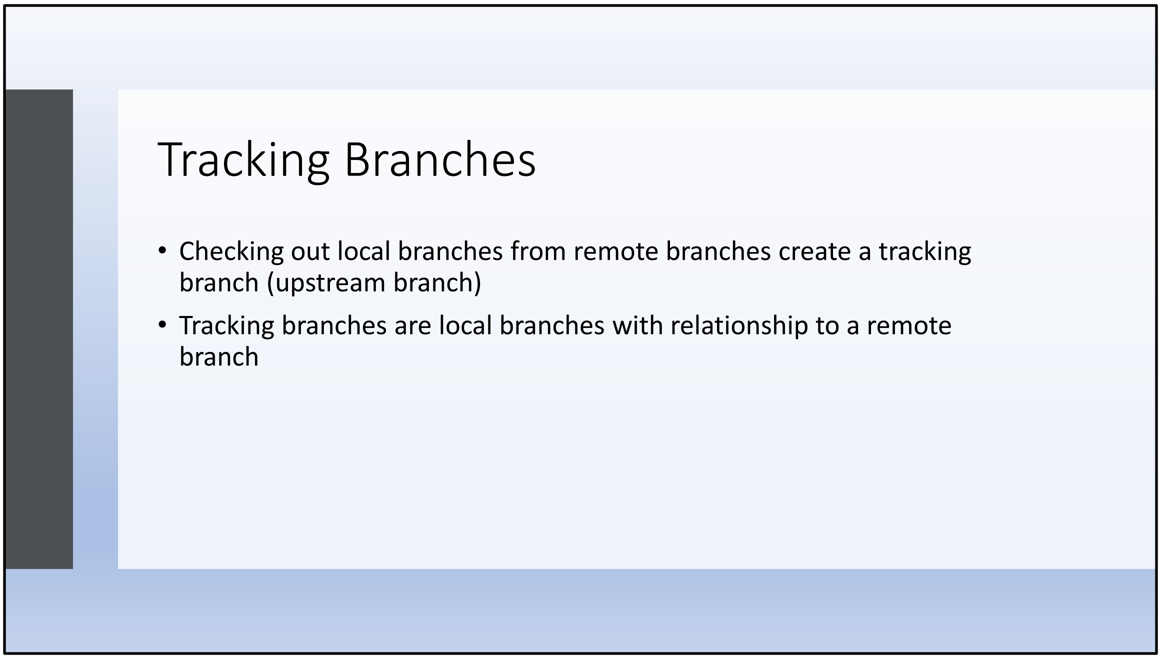
topic

main



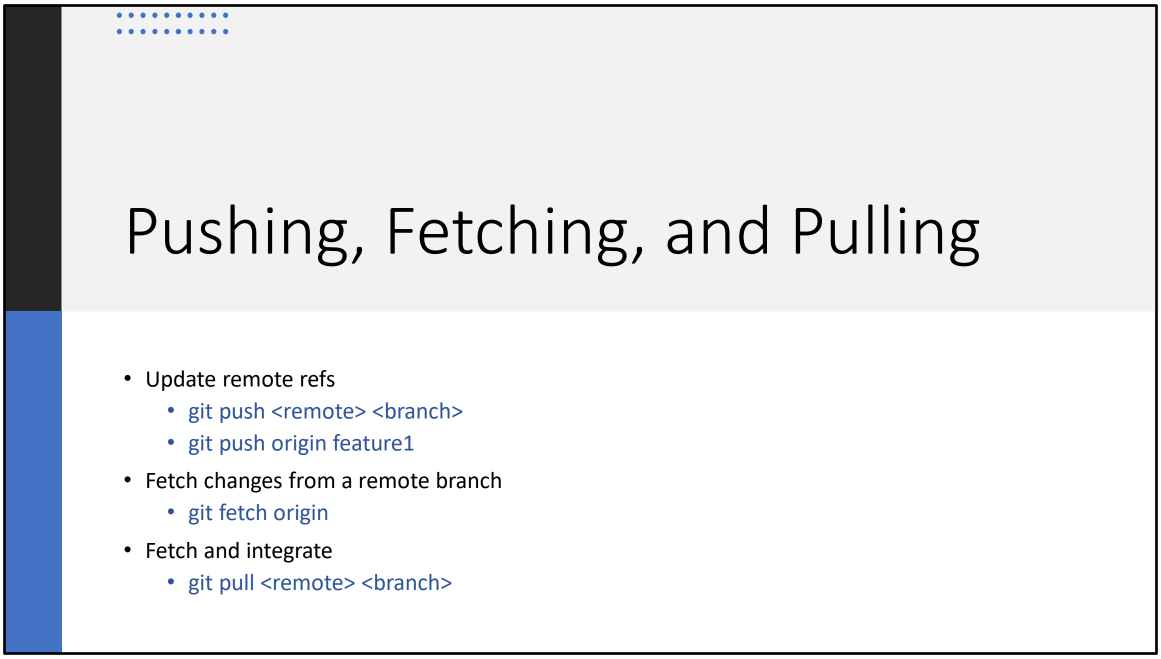
Remote Branches

- Remote branches are references to the state of branches in your remote repositories.
- Remote branches acts as bookmarks to remind you on the status of remote repositories.



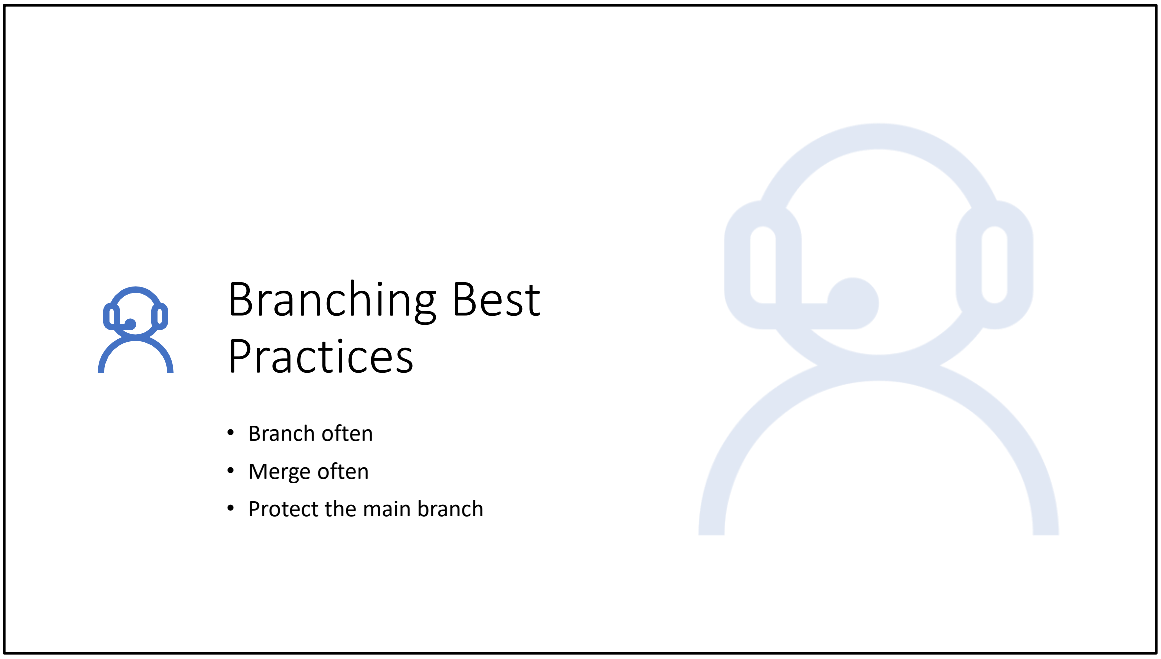
Tracking Branches

- Checking out local branches from remote branches create a tracking branch (upstream branch)
- Tracking branches are local branches with relationship to a remote branch



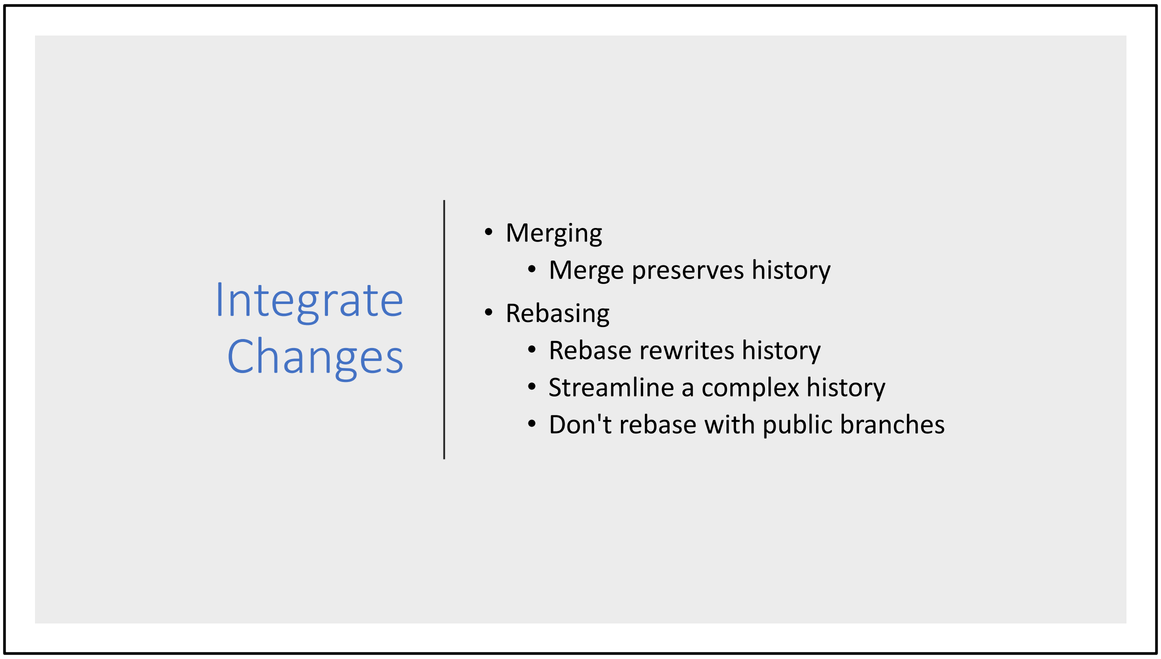
Pushing, Fetching, and Pulling

- Update remote refs
  - git push <remote> <branch>
  - git push origin feature1
- Fetch changes from a remote branch
  - git fetch origin
- Fetch and integrate
  - git pull <remote> <branch>



Branching Best Practices

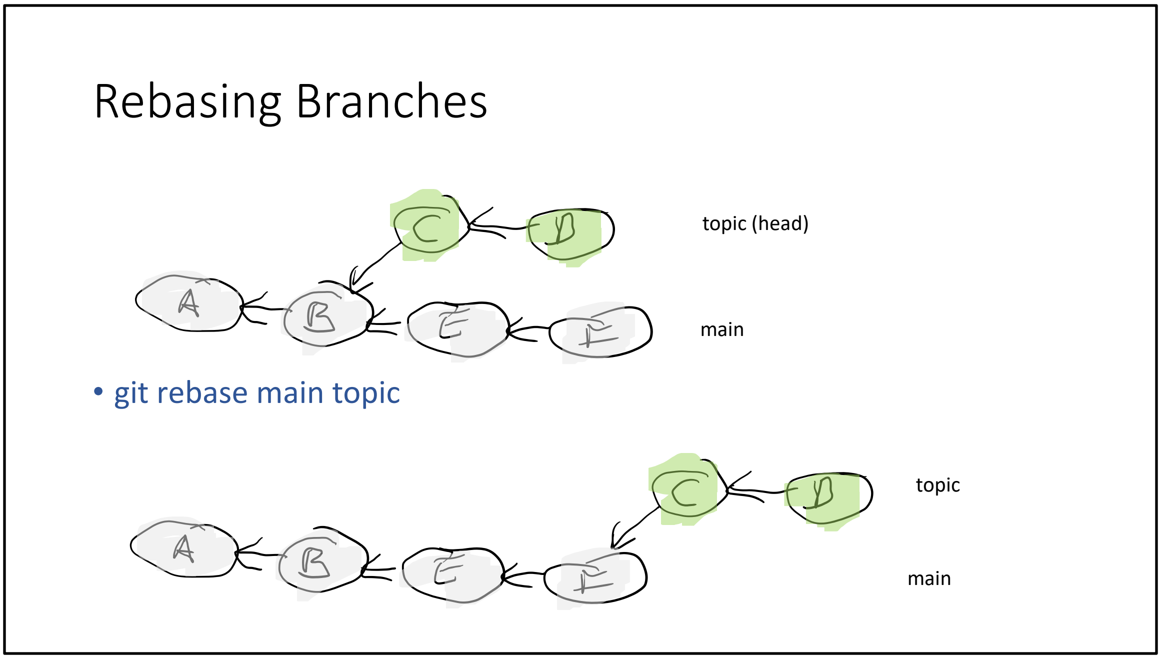
- Branch often
- Merge often
- Protect the main branch



Integrate Changes

- Merging
  - Merge preserves history
- Rebasing
  - Rebase rewrites history
  - Streamline a complex history
  - Don't rebase with public branches

```
git branch –set-upstream-to=featurebranch1 main
```



Rebasing Branches

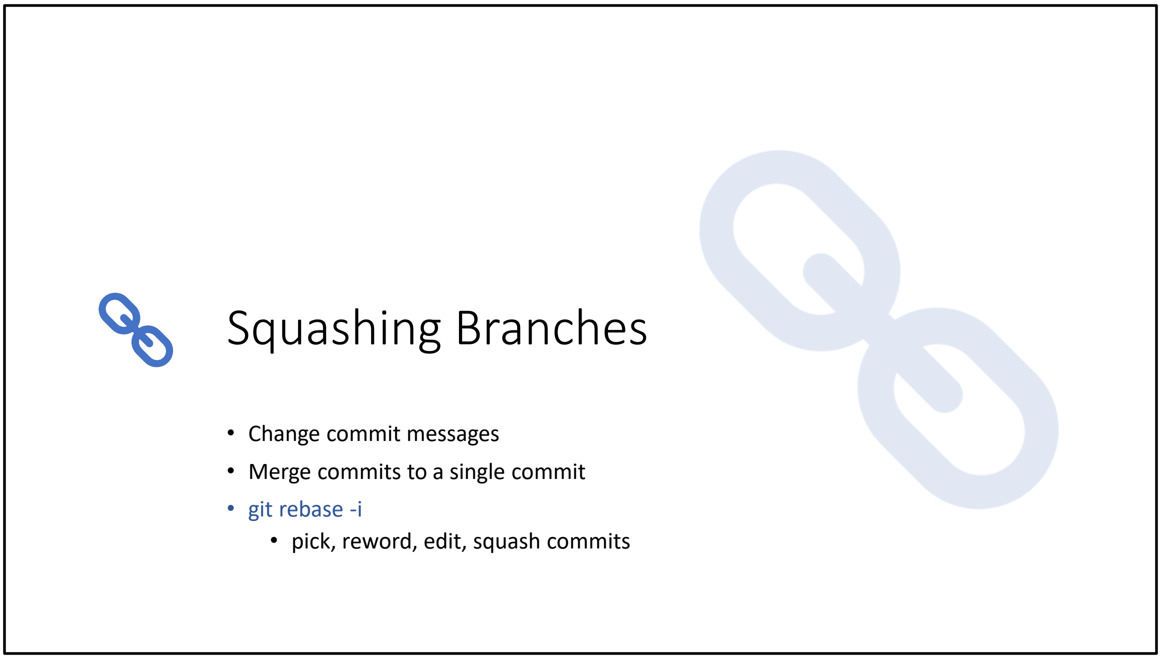
topic (head)

main

- git rebase main topic

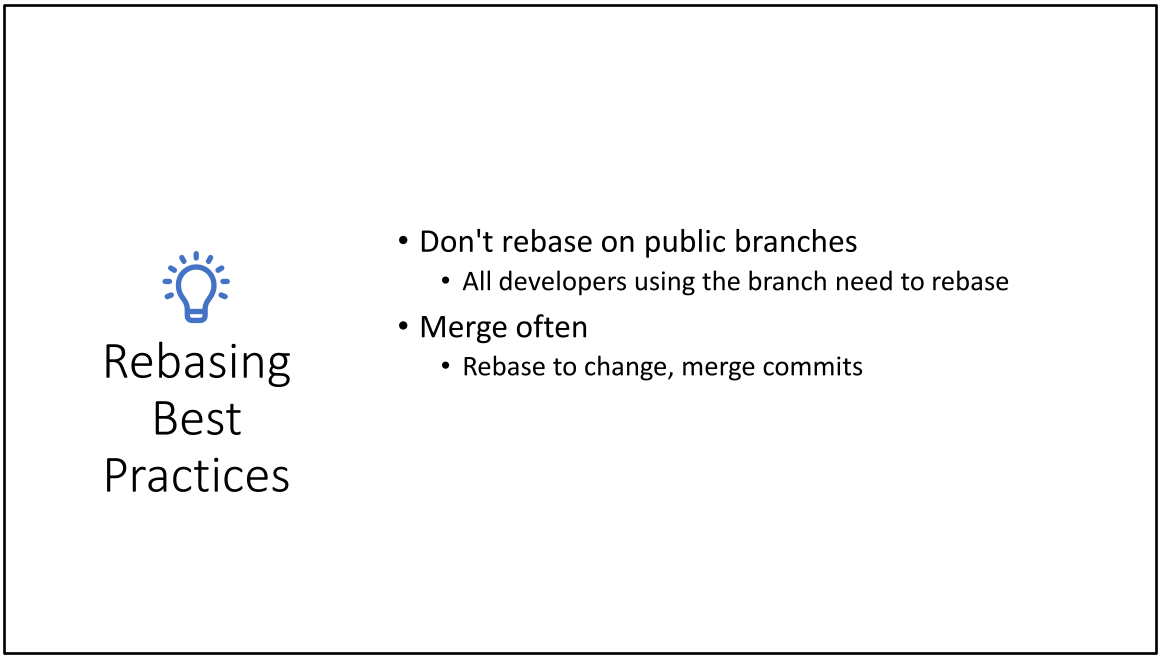
topic

main



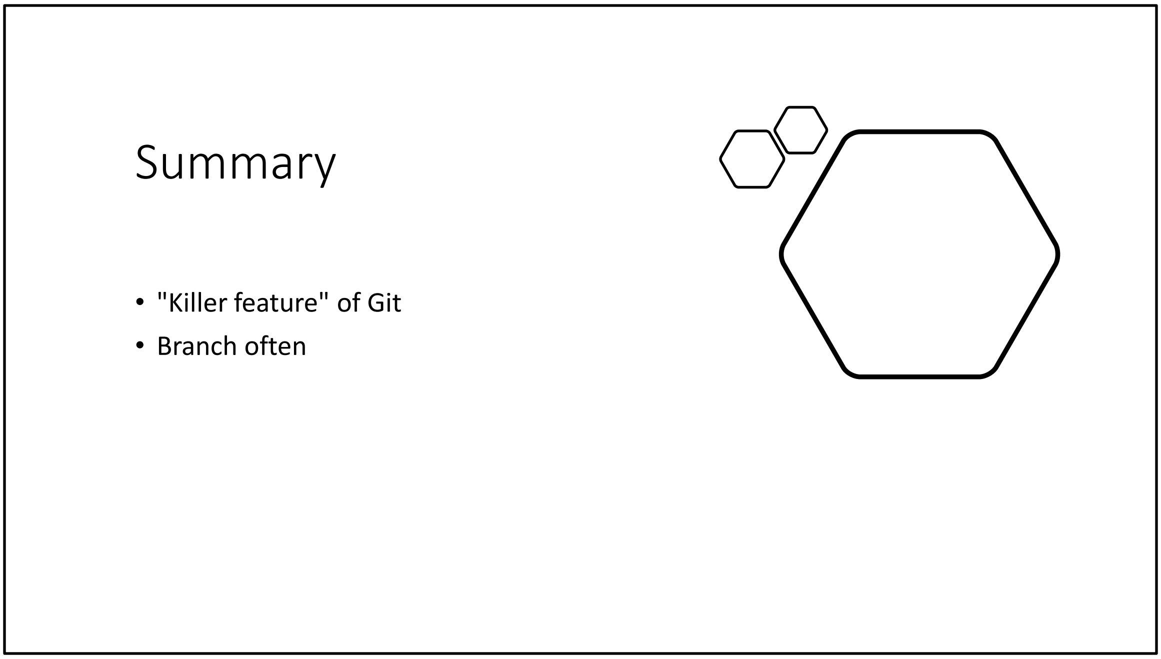
Squashing Branches

- Change commit messages
- Merge commits to a single commit
- git rebase -i
  - pick, reword, edit, squash commits



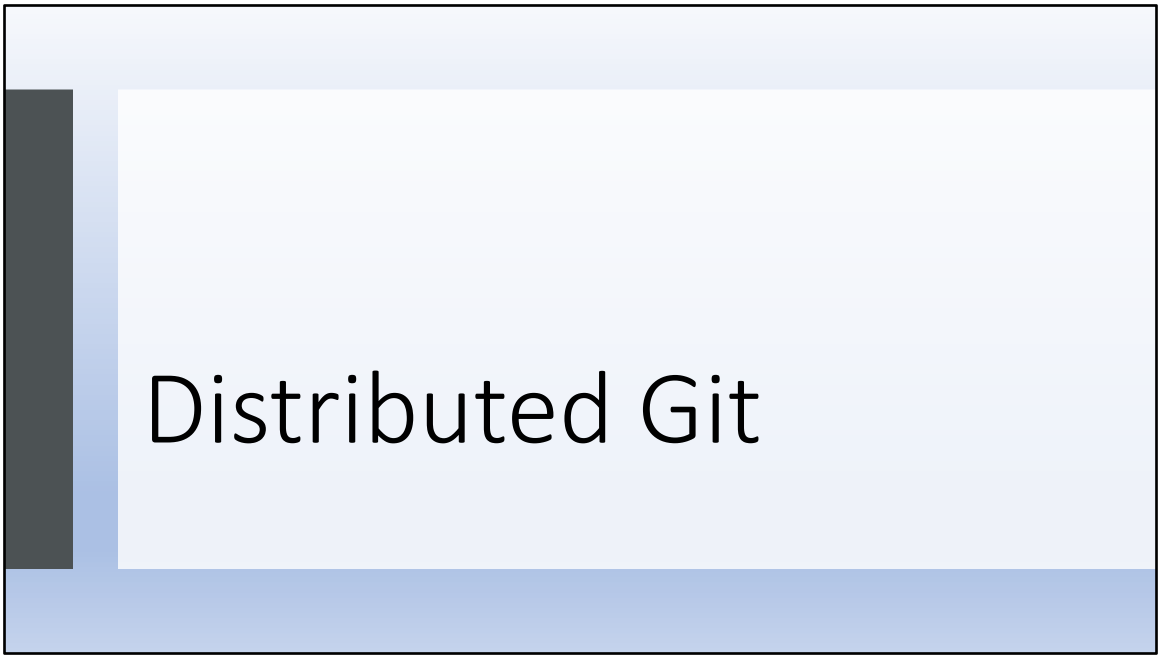
Rebasing Best Practices

- Don't rebase on public branches
  - All developers using the branch need to rebase
- Merge often
  - Rebase to change, merge commits

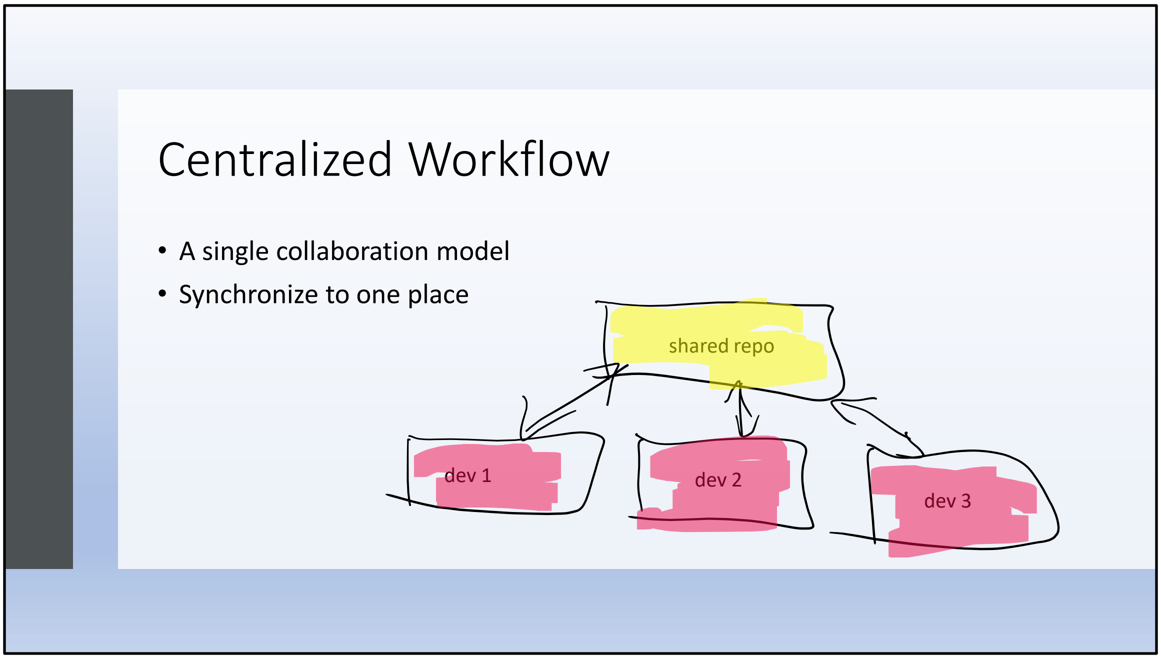


Summary

- "Killer feature" of Git
- Branch often



Distributed Git



Centralized Workflow

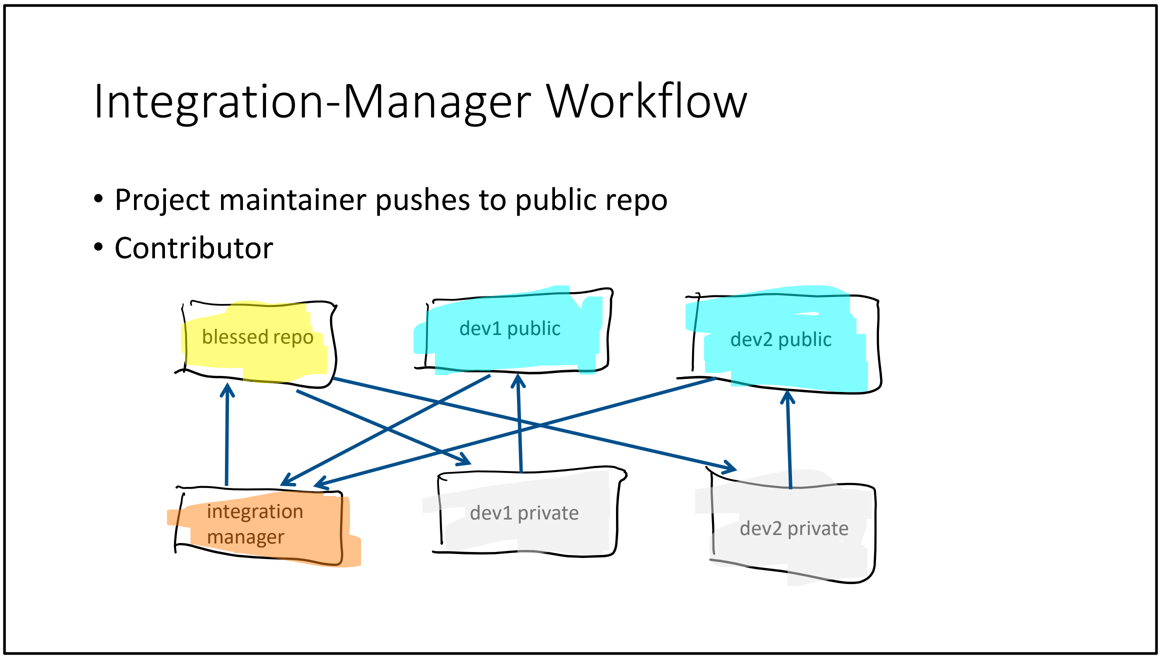
- A single collaboration model
- Synchronize to one place

shared repo

dev 1

dev 2

dev 3



Integration-Manager Workflow

- Project maintainer pushes to public repo
- Contributor

blessed repo

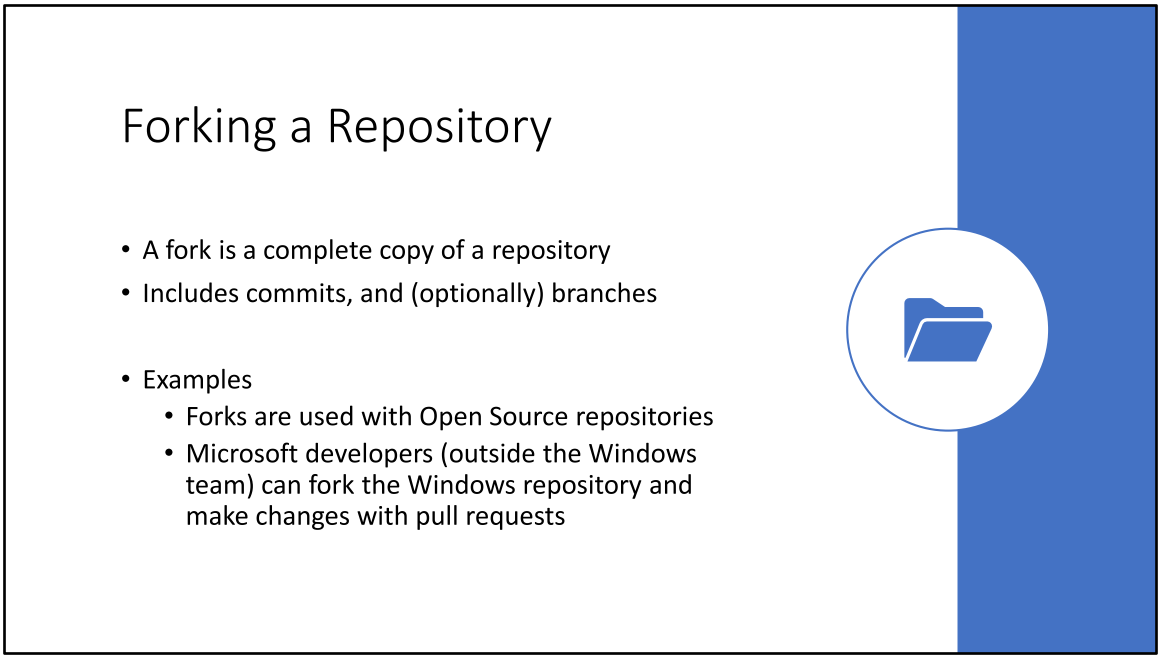
dev1 public

dev2 public

integration manager

dev1 private

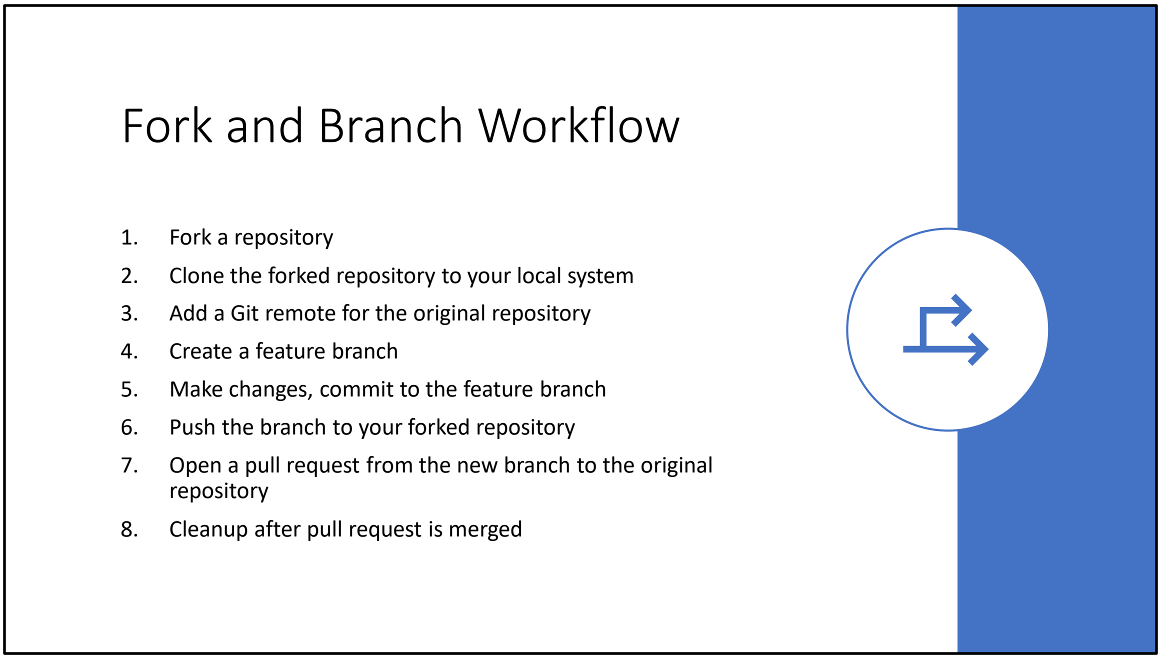
dev2 private



Forking a Repository

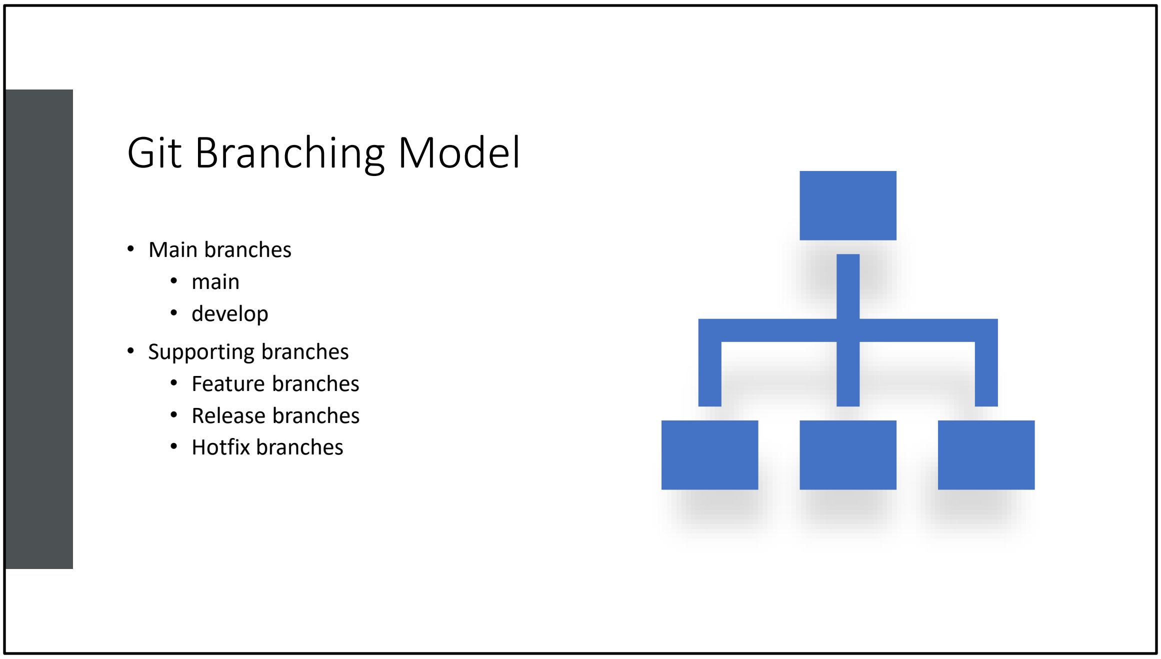
- A fork is a complete copy of a repository
- Includes commits, and (optionally) branches

- Examples
  - Forks are used with Open Source repositories
  - Microsoft developers (outside the Windows team) can fork the Windows repository and make changes with pull requests



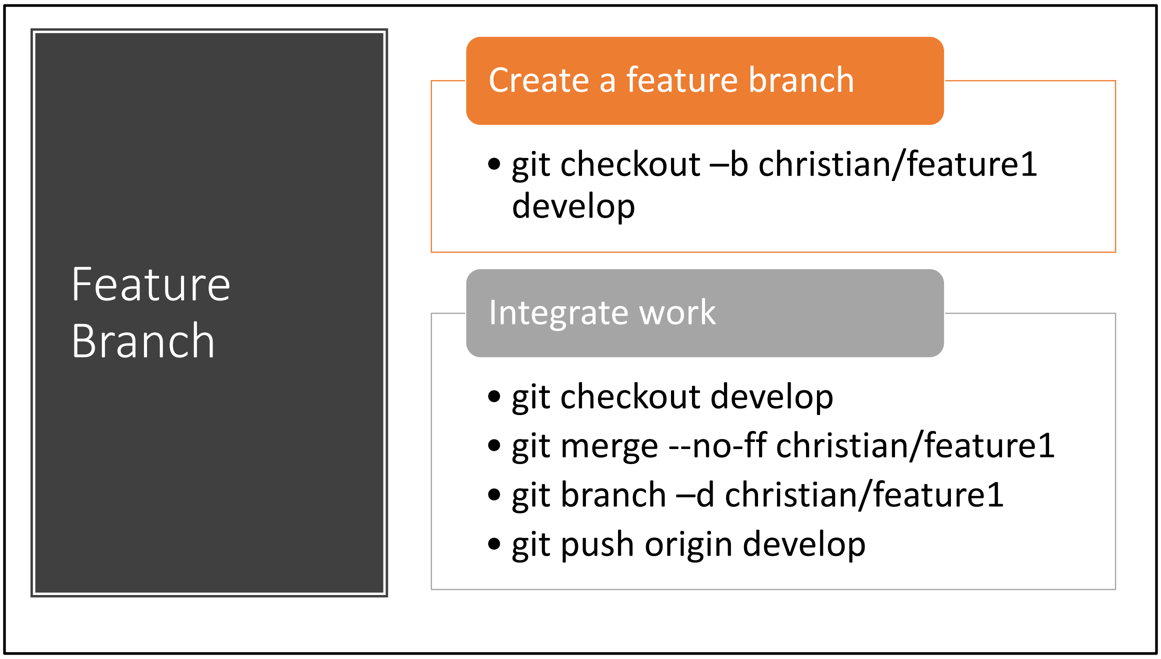
Fork and Branch Workflow

1. Fork a repository
2. Clone the forked repository to your local system
3. Add a Git remote for the original repository
4. Create a feature branch
5. Make changes, commit to the feature branch
6. Push the branch to your forked repository
7. Open a pull request from the new branch to the original repository
8. Cleanup after pull request is merged



Git Branching Model

- Main branches
  - main
  - develop
- Supporting branches
  - Feature branches
  - Release branches
  - Hotfix branches



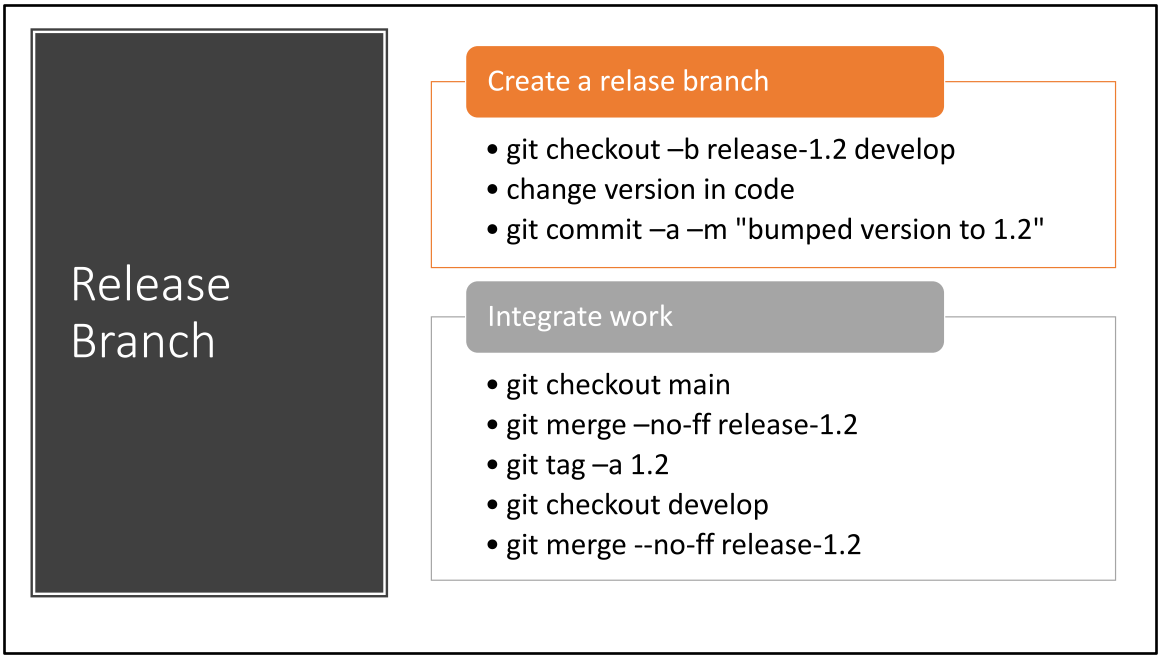
Feature Branch

Create a feature branch

- git checkout –b christian/feature1 develop

Integrate work

- git checkout develop
- git merge --no-ff christian/feature1
- git branch –d christian/feature1
- git push origin develop



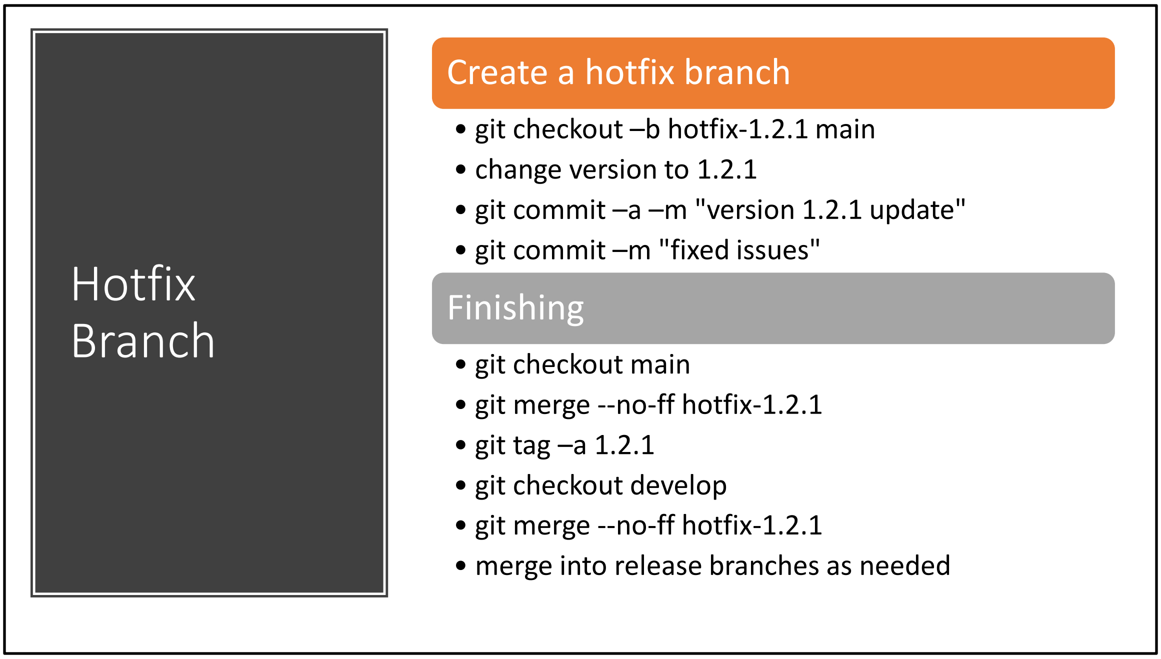
Release Branch

Create a relase branch

- git checkout –b release-1.2 develop
- change version in code
- git commit –a –m "bumped version to 1.2"

Integrate work

- git checkout main
- git merge –no-ff release-1.2
- git tag –a 1.2
- git checkout develop
- git merge --no-ff release-1.2



Hotfix Branch

Create a hotfix branch

- git checkout –b hotfix-1.2.1 main
- change version to 1.2.1
- git commit –a –m "version 1.2.1 update"
- git commit –m "fixed issues"

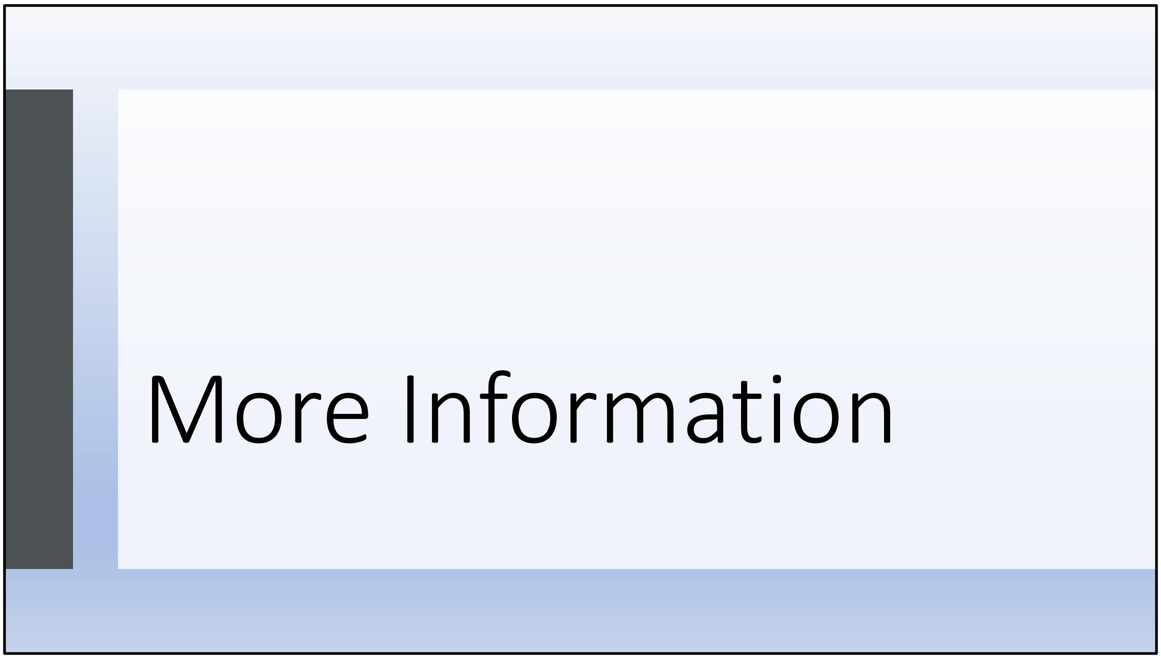
Finishing

- git checkout main
- git merge --no-ff hotfix-1.2.1
- git tag –a 1.2.1
- git checkout develop
- git merge --no-ff hotfix-1.2.1
- merge into release branches as needed

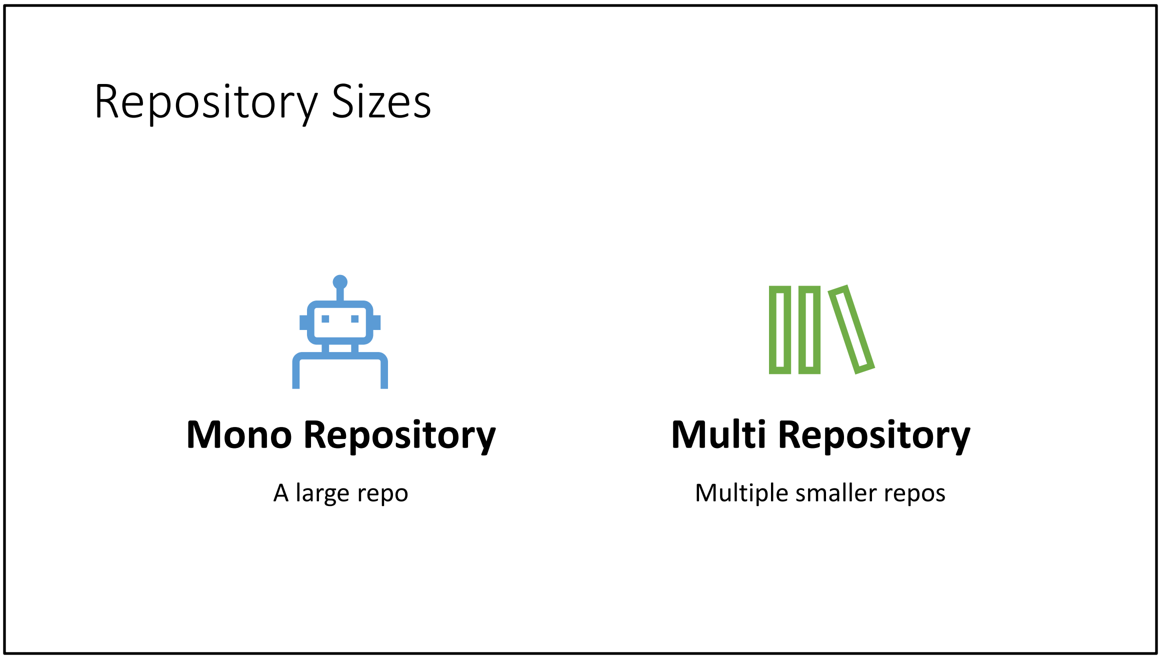
Summary

Git allows flexible workflows

Depending on your team size and culture define a practical workflow to use



More Information



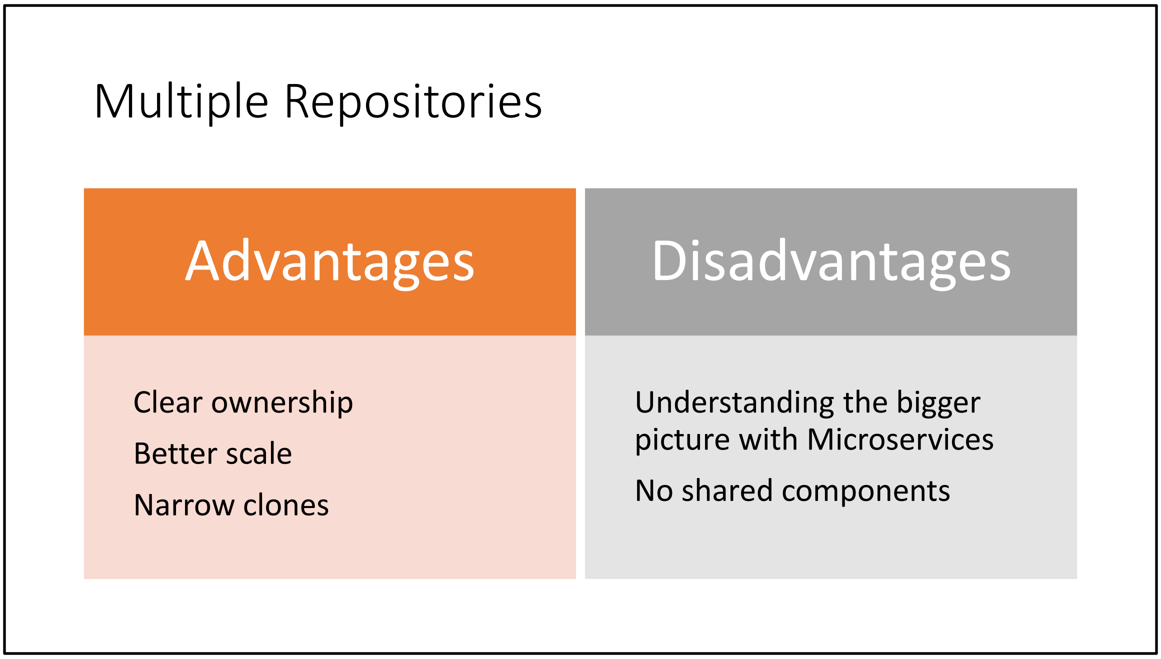
Repository Sizes

Mono Repository

A large repo

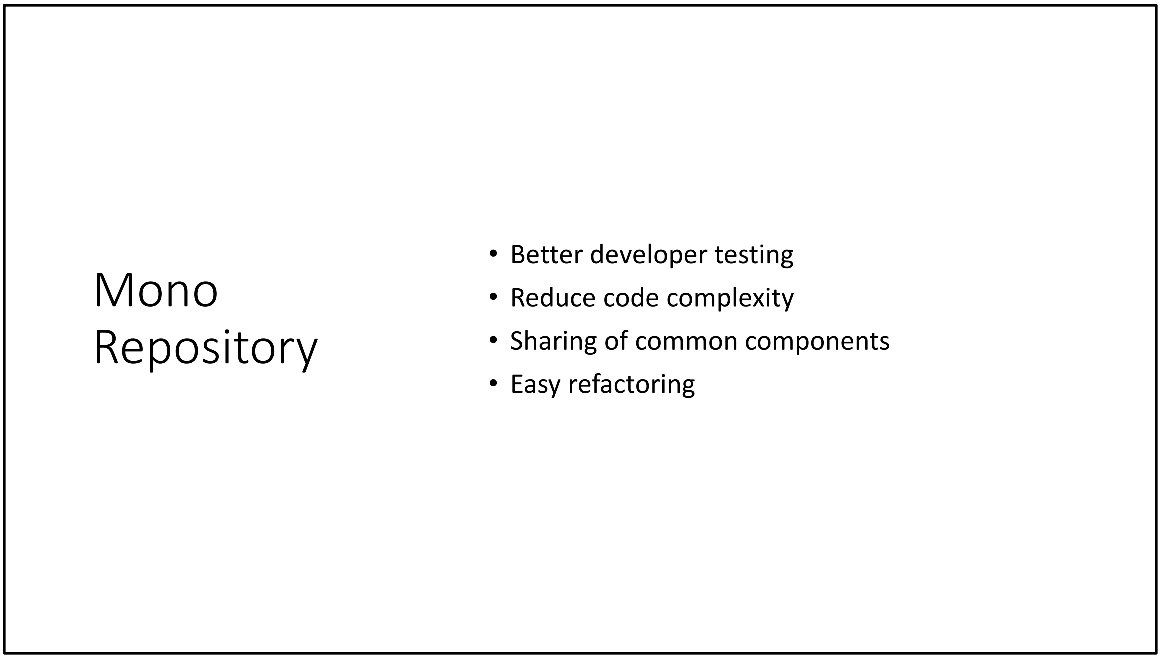
Multi Repository

Multiple smaller repos



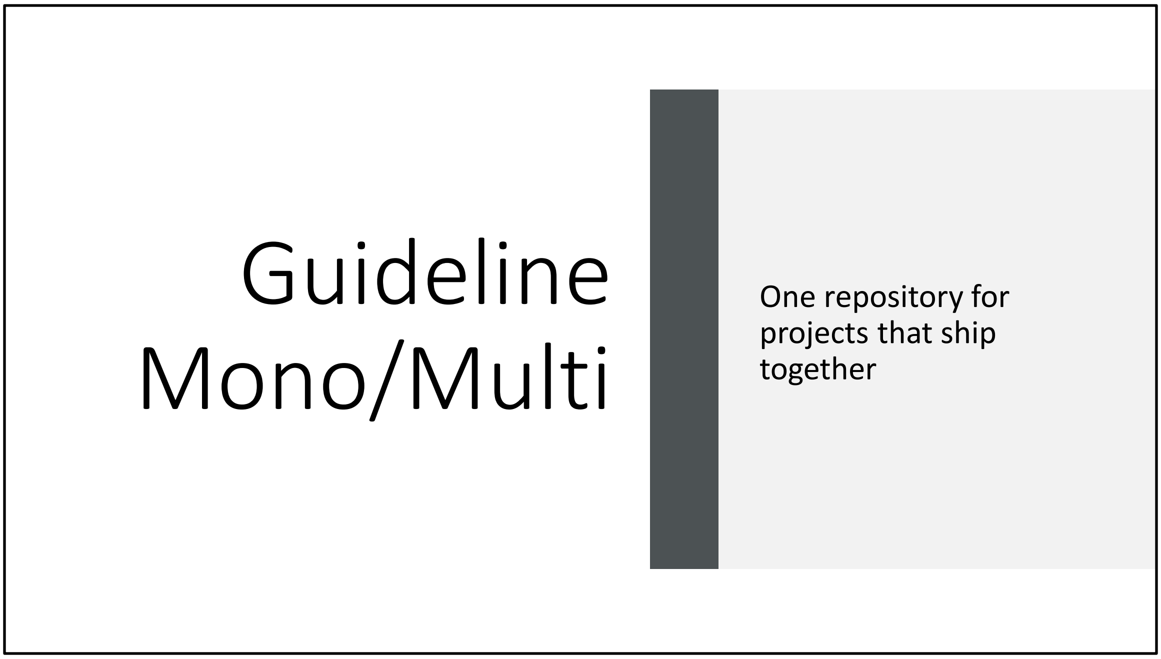
Multiple Repositories

Advantages	Disadvantages
Clear ownership Better scale Narrow clones	Understanding the bigger picture with Microservices No shared components



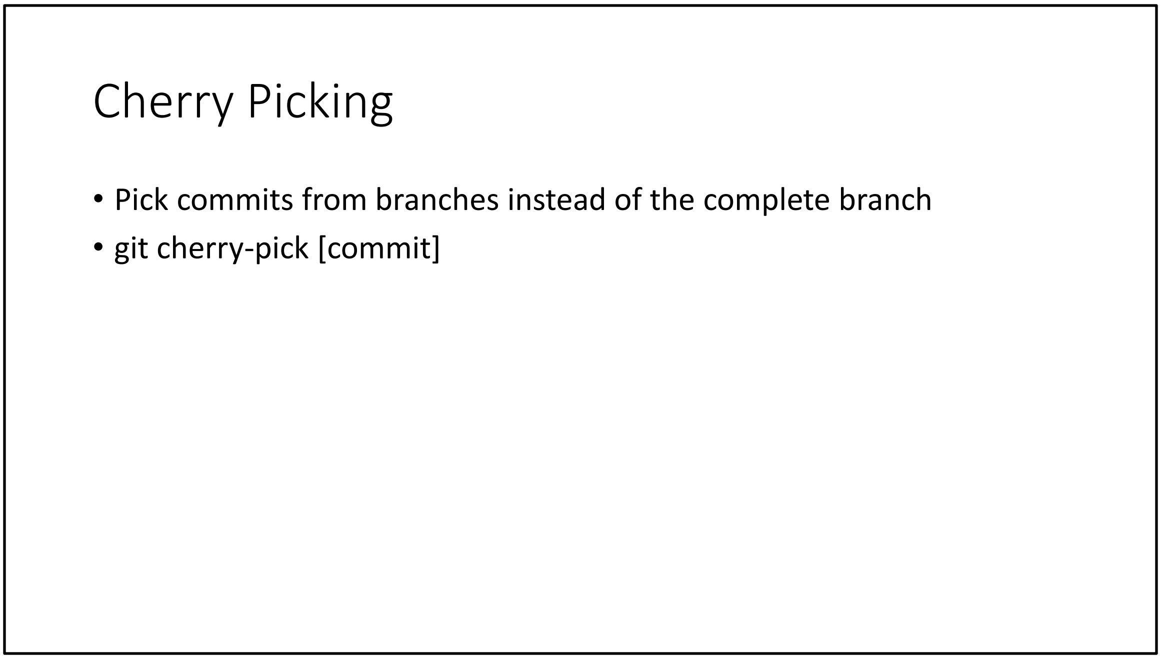
Mono Repository

- Better developer testing
- Reduce code complexity
- Sharing of common components
- Easy refactoring



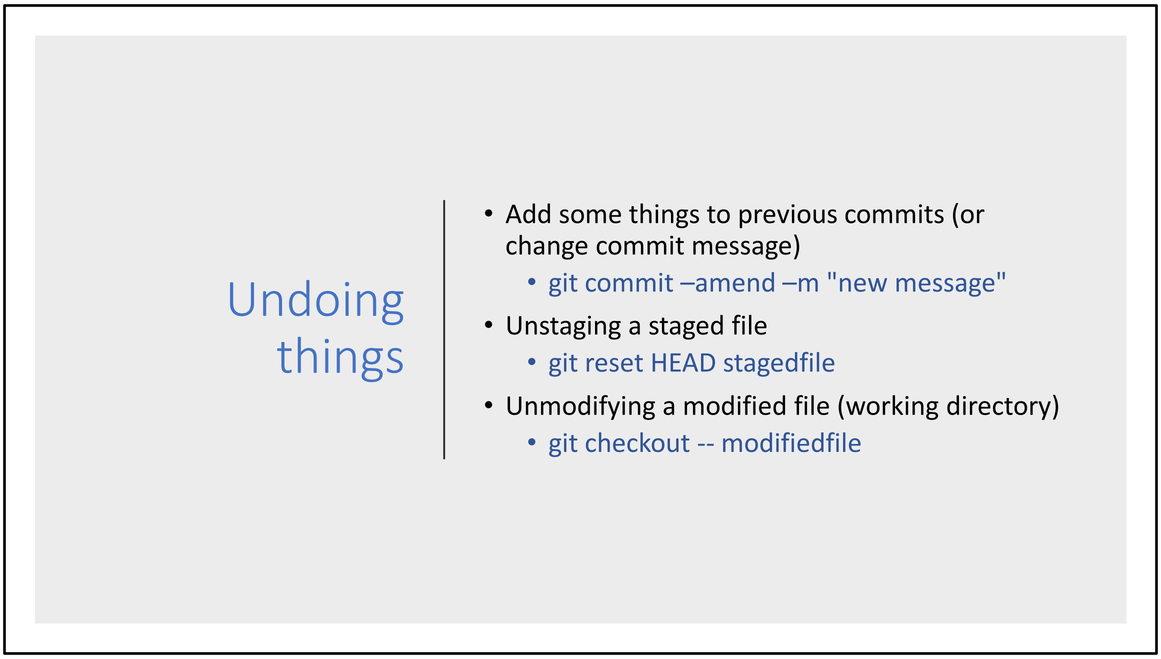
Guideline Mono/Multi

One repository for projects that ship together



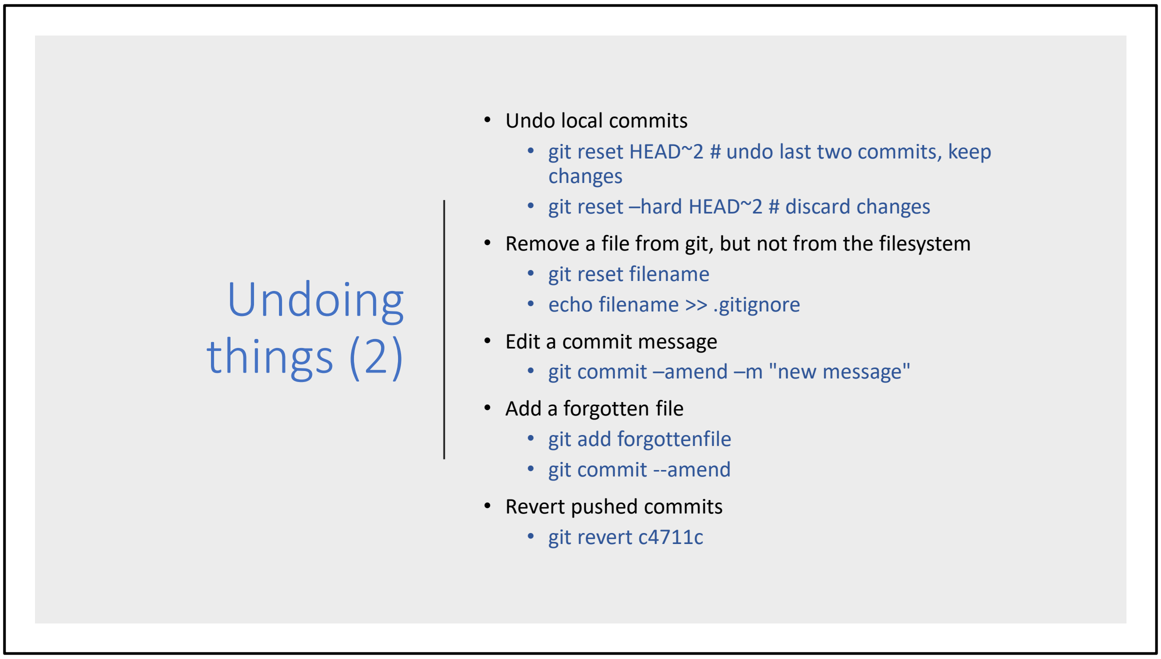
Cherry Picking

- Pick commits from branches instead of the complete branch
- git cherry-pick [commit]



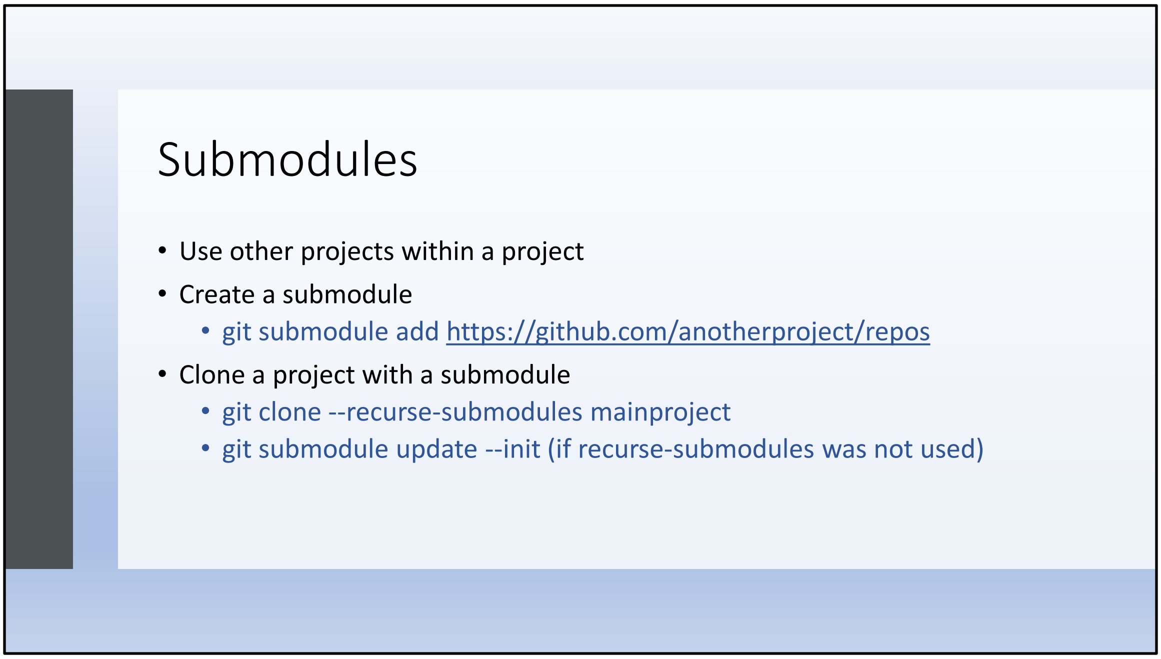
Undoing things

- Add some things to previous commits (or change commit message)
  - git commit –amend –m "new message"
- Unstaging a staged file
  - git reset HEAD stagedfile
- Unmodifying a modified file (working directory)
  - git checkout -- modifiedfile



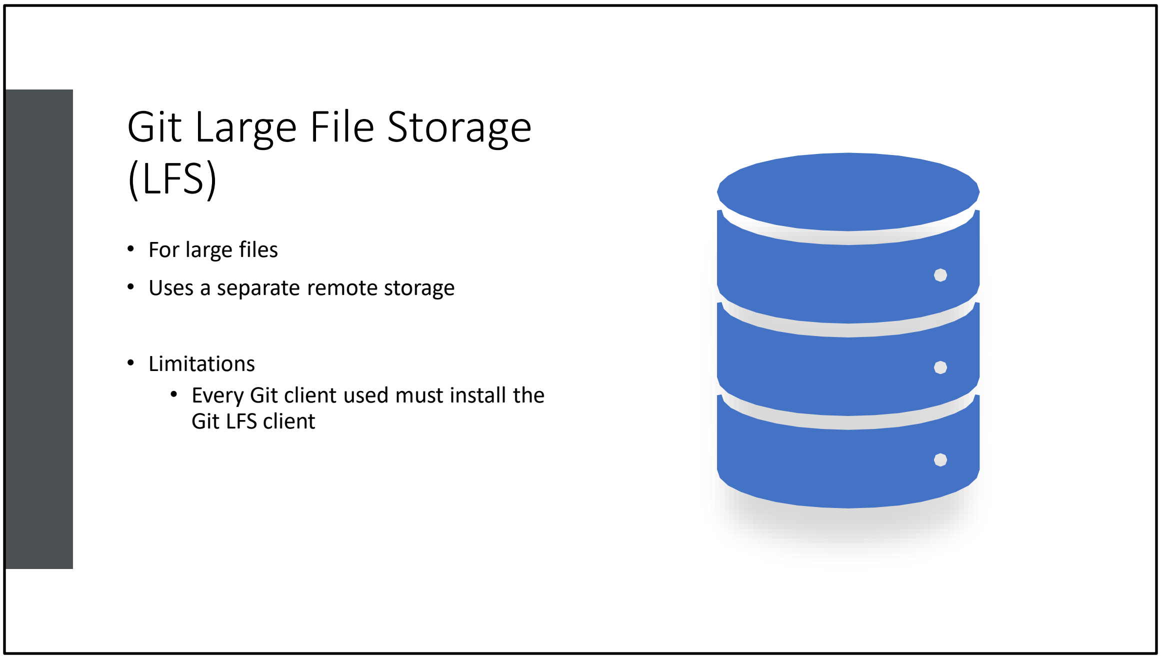
Undoing things (2)

- Undo local commits
  - git reset HEAD~2 # undo last two commits, keep changes
  - git reset –hard HEAD~2 # discard changes
- Remove a file from git, but not from the filesystem
  - git reset filename
  - echo filename >> .gitignore
- Edit a commit message
  - git commit –amend –m "new message"
- Add a forgotten file
  - git add forgottenfile
  - git commit --amend
- Revert pushed commits
  - git revert c4711c



Submodules

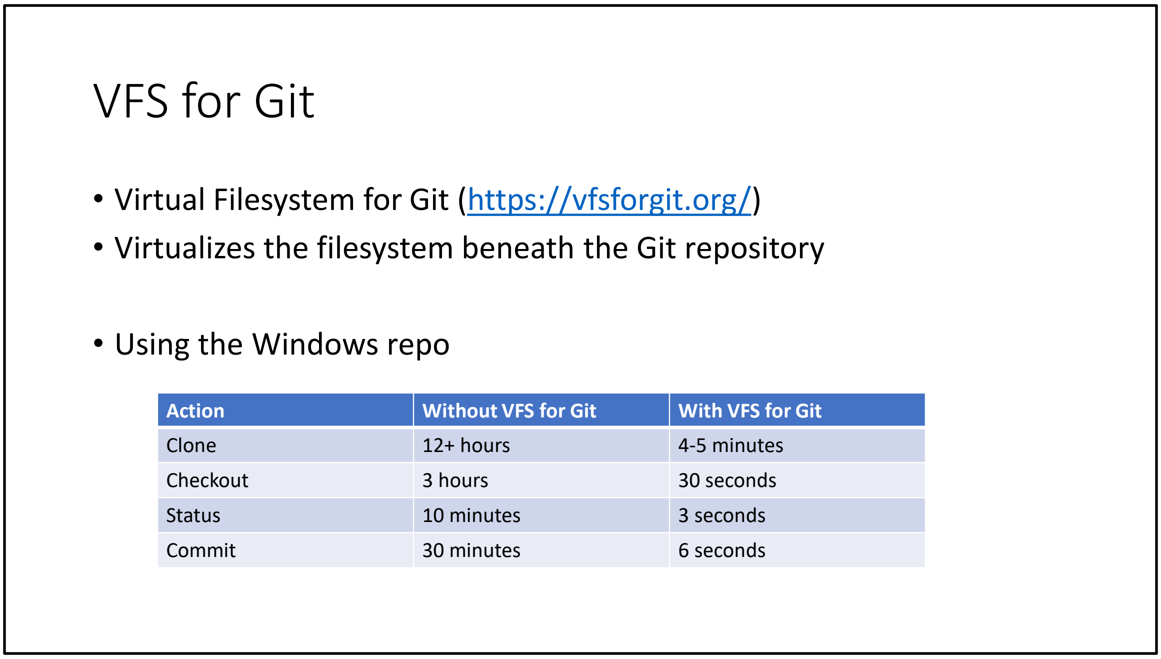
- Use other projects within a project
- Create a submodule
  - git submodule add https://github.com/anotherproject/repos
- Clone a project with a submodule
  - git clone --recurse-submodules mainproject
  - git submodule update --init (if recurse-submodules was not used)



Git Large File Storage (LFS)

- For large files
- Uses a separate remote storage

- Limitations
  - Every Git client used must install the Git LFS client

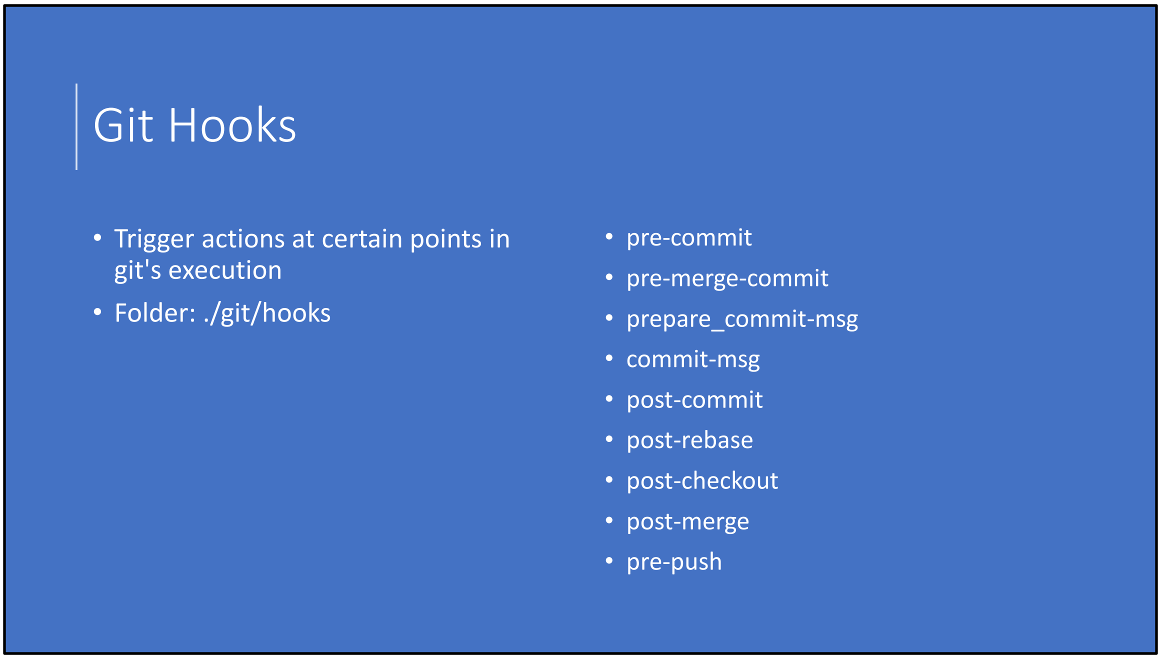


VFS for Git

- Virtual Filesystem for Git (https://vfsforgit.org/)
- Virtualizes the filesystem beneath the Git repository

- Using the Windows repo

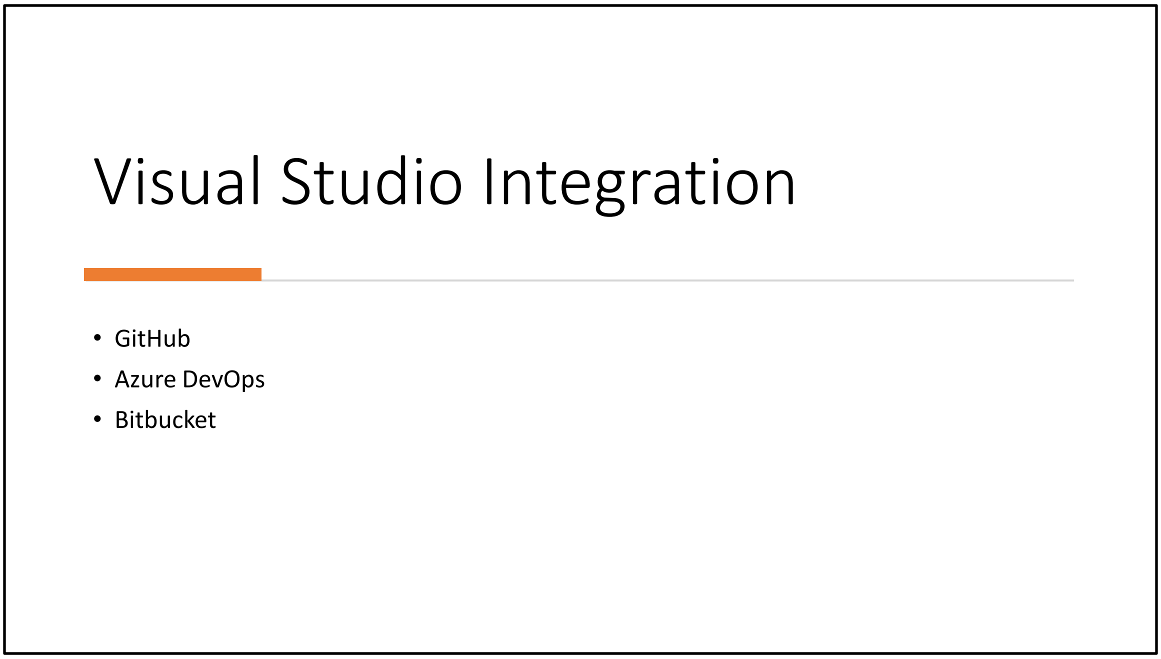
Action	Without VFS for Git	With VFS for Git
Clone	12+ hours	4-5 minutes
Checkout	3 hours	30 seconds
Status	10 minutes	3 seconds
Commit	30 minutes	6 seconds



Git Hooks

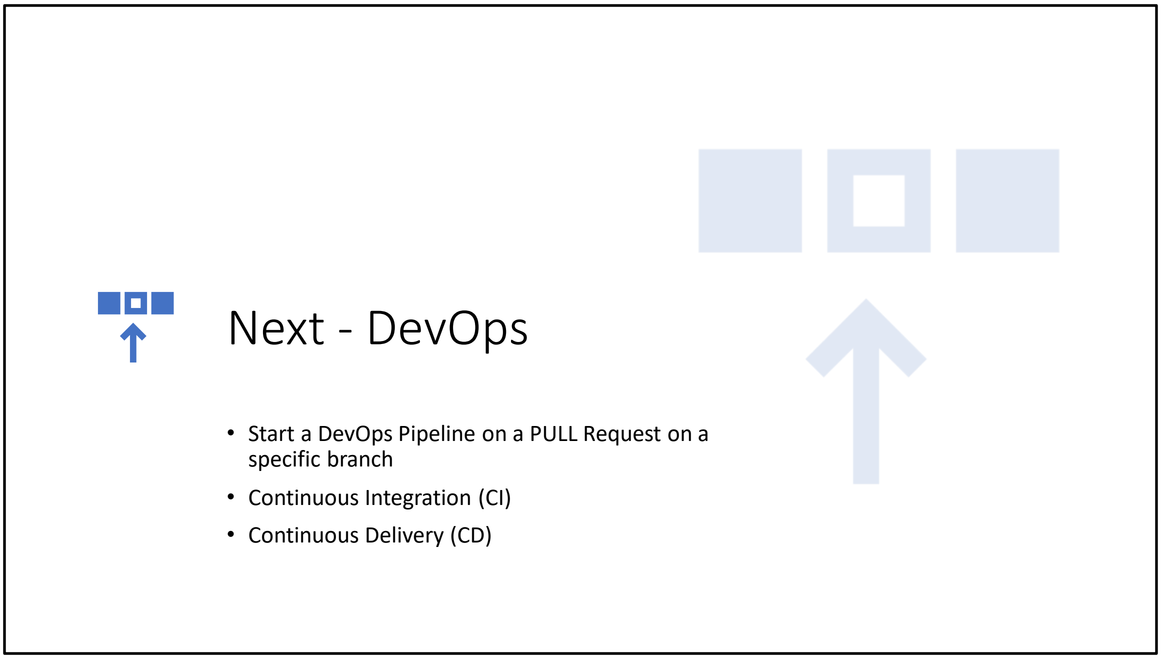
- Trigger actions at certain points in git's execution
- Folder: ./git/hooks

- pre-commit
- pre-merge-commit
- prepare_commit-msg
- commit-msg
- post-commit
- post-rebase
- post-checkout
- post-merge
- pre-push



Visual Studio Integration

- GitHub
- Azure DevOps
- Bitbucket



Next - DevOps

- Start a DevOps Pipeline on a PULL Request on a specific branch
- Continuous Integration (CI)
- Continuous Delivery (CD)

Summary

- Git
- Commits
- Branches
- Distributed Git
- Workflows