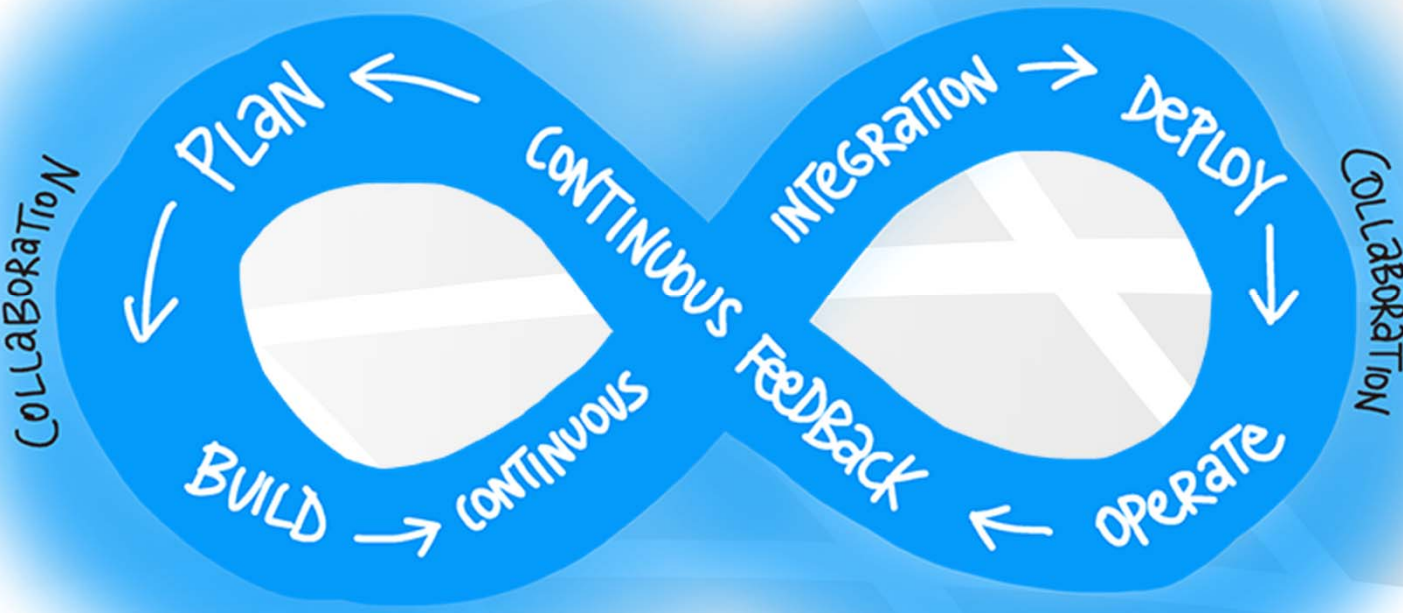
Introuction to devops

Dan Morgenstern

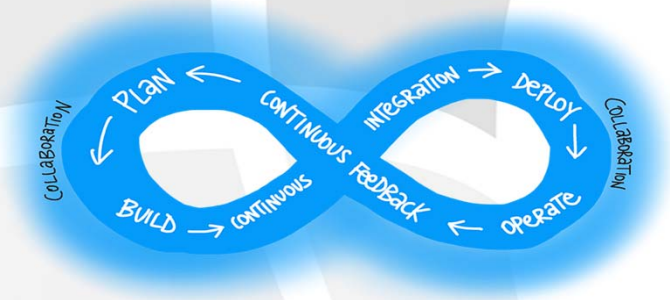
Agenda

- What is devops?
- Why devops?
- Devops elements
- Devops course

What is devops?

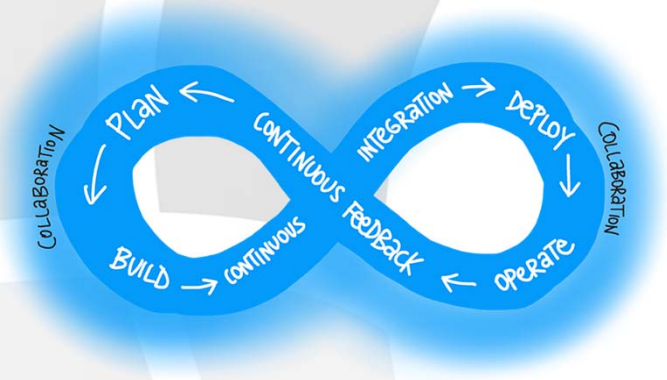


What is devops?



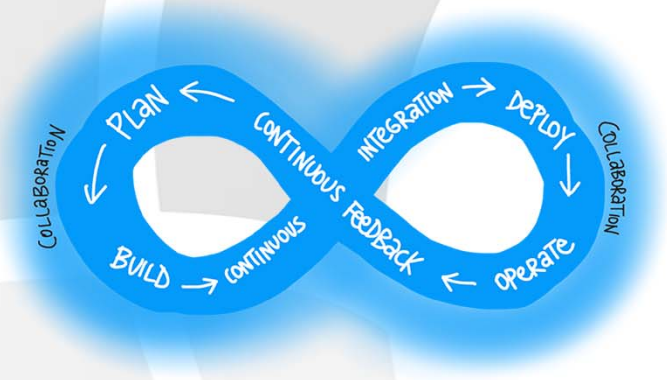
- Development + operations
- the union of people, process, and products to enable continuous delivery of value to our end users.
- includes agile planning, continuous integration, continuous delivery, and monitoring of applications

Why devops?

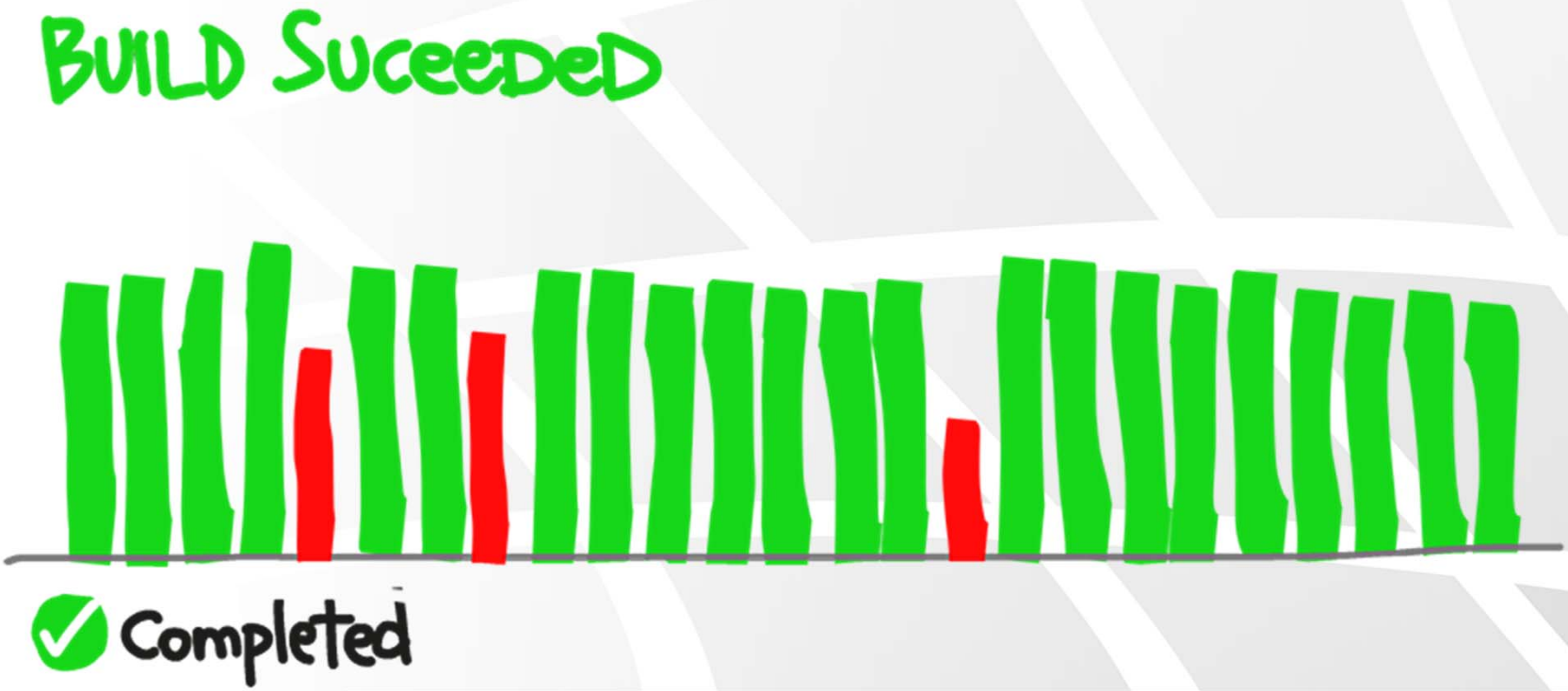


- SPEED TO MARKET
  - Shipping more quickly
  - Adopting new technology more quickly
  - Quicker implementation of new features
- Better and quicker response to customer found bugs
- Improved quality by automated testing

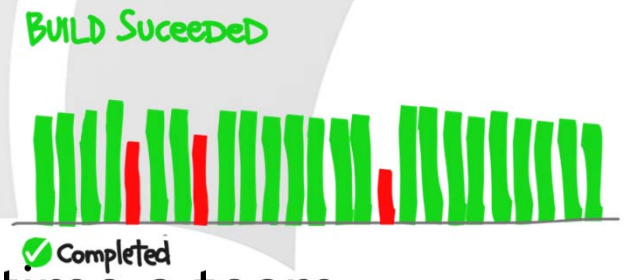
Devops elements



- Continuous Integration (CI)

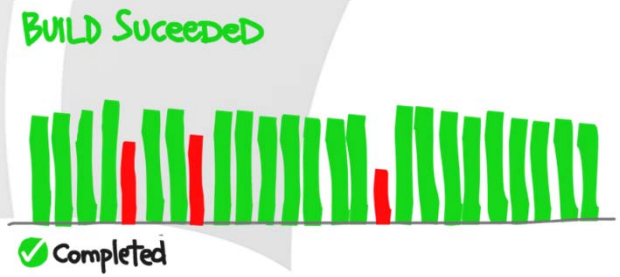


Devops elements - CI



- Automating the build and testing of code every time a team member commits changes to version control
- Triggering build system to grab the latest code from the shared repository and to build, test, and validate the full master branch
- Developers work isolated and CI integrates their changes with the rest of the team member's changes

Devops elements - CI

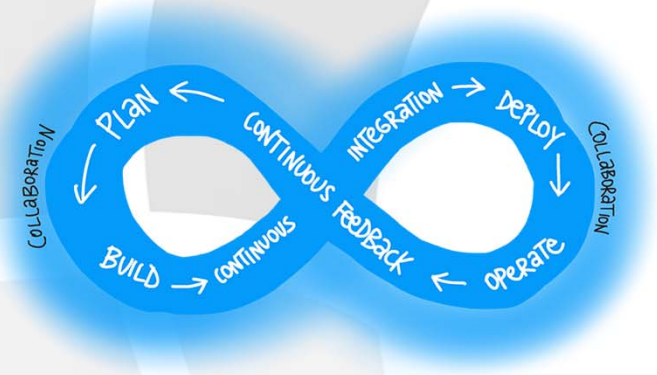


- CI keeps the master branch clean.

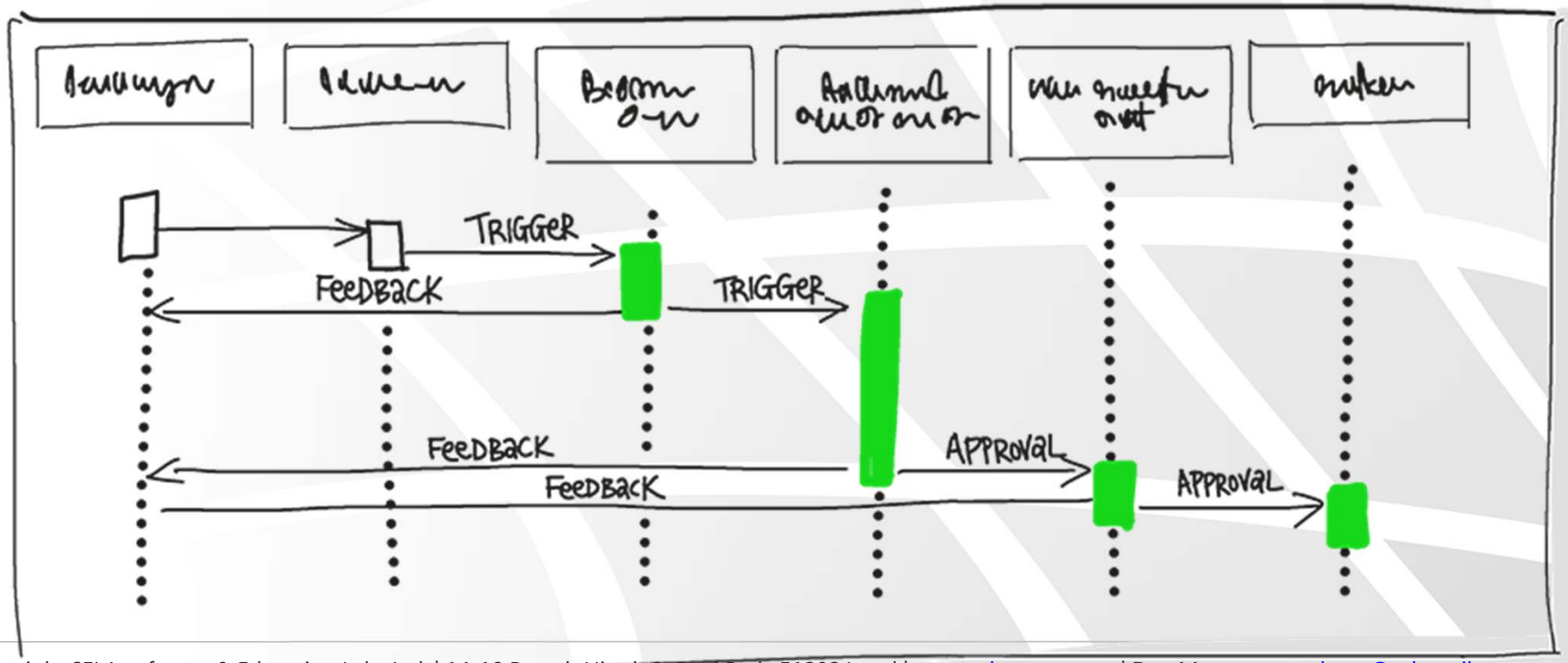
  changes get merged into the master branch after feature is complete and approved.

- CI ensures bugs are caught earlier in the development cycle, which makes them less expensive to fix.

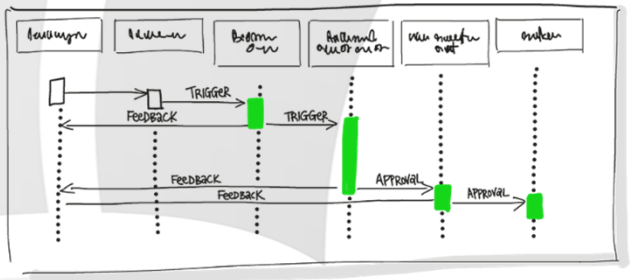
Devops elements



- Continuous Delivery (CD)



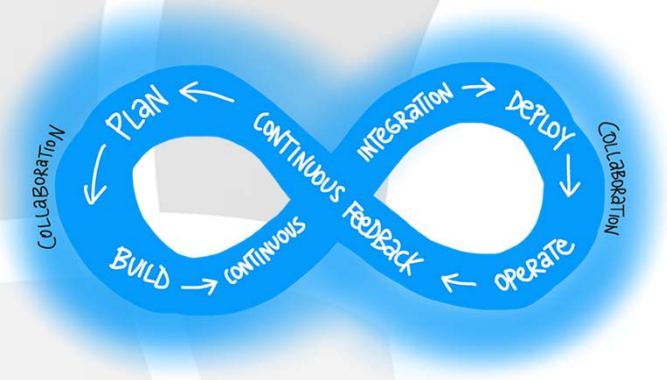
Devops elements - CD



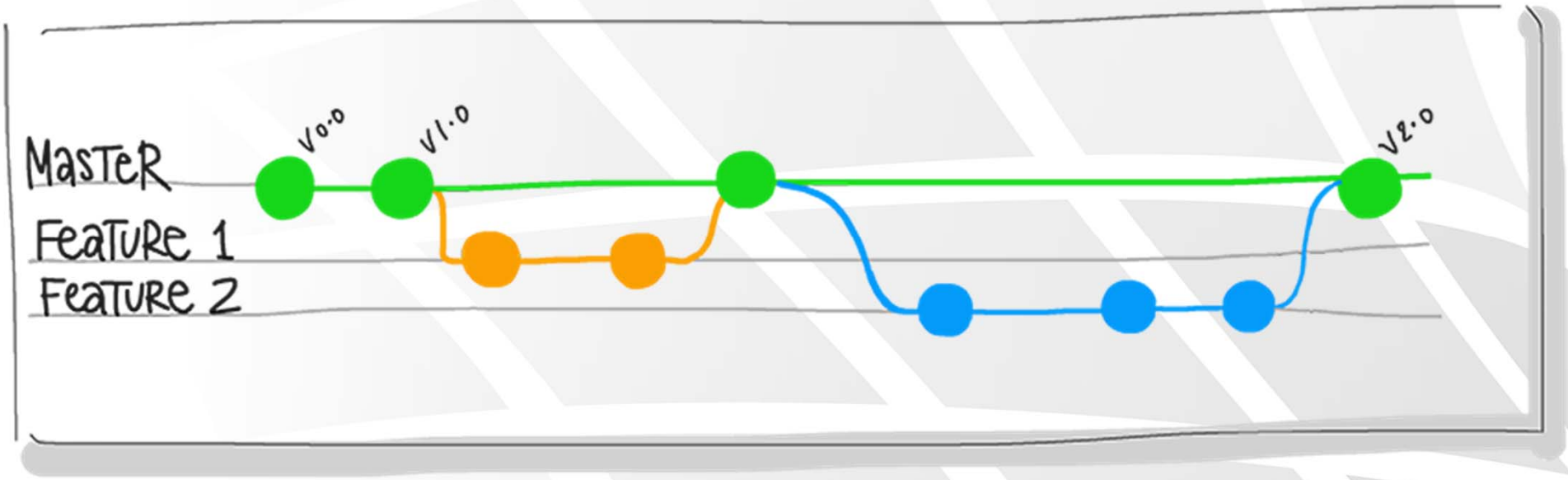
- CD is the process to build, test, configure and deploy to a production environment. Multiple testing or staging environments create a Release Pipeline.

- Without CD, software release cycles were previously the bottleneck . Manual processes led to unreliable releases that produced delays and errors.

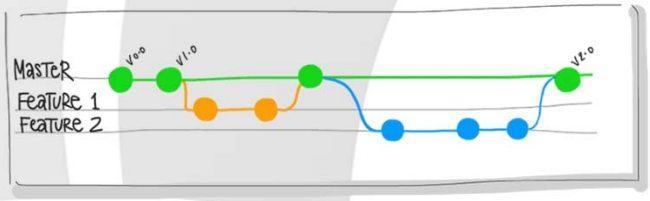
Devops elements



- Version Control (usually git)

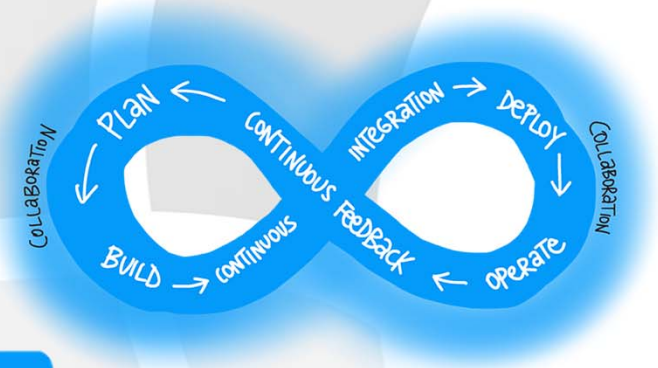


Devops elements - git

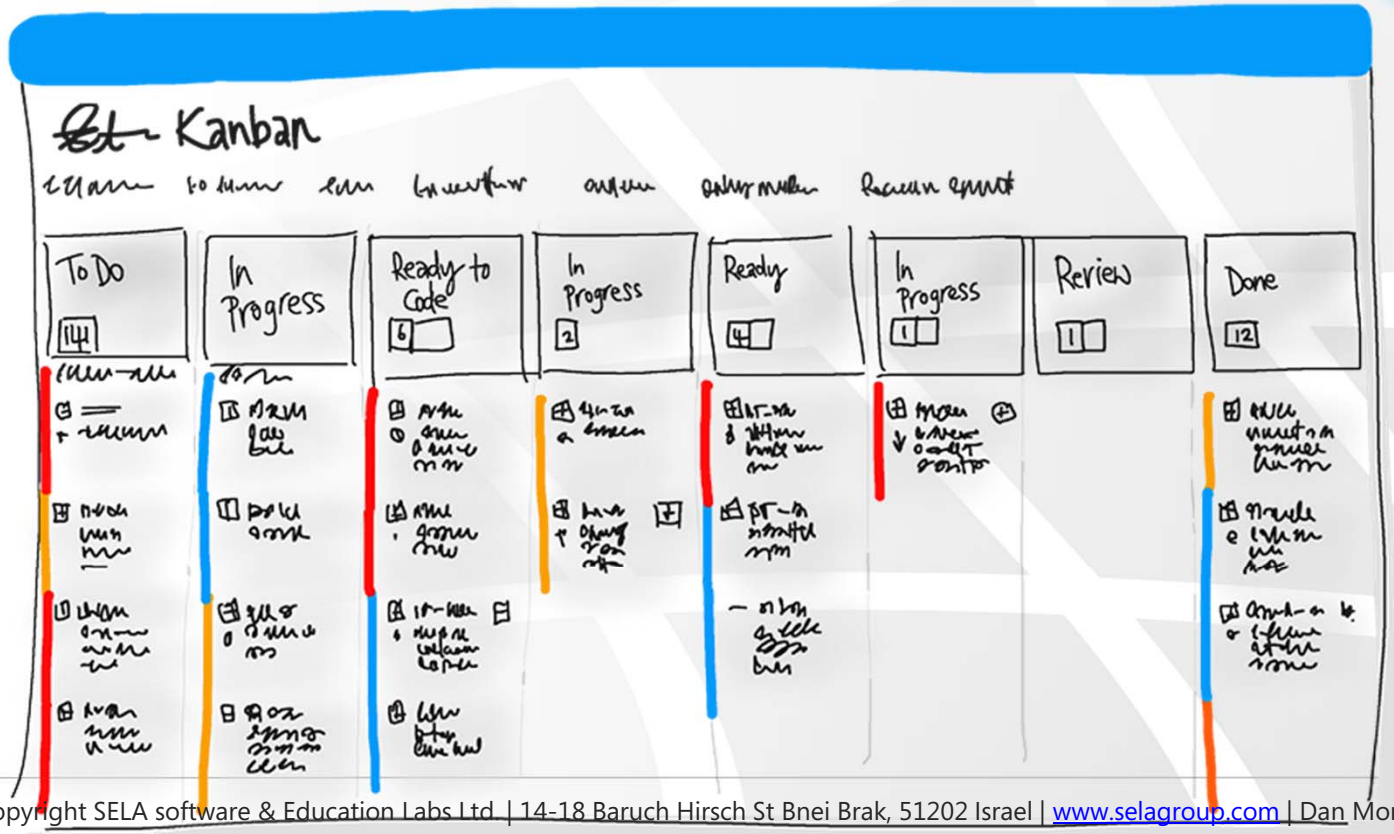


- the most commonly used version control system today
- distributed version control system
- Branches - lightweight pointers to work in progress
- Pull requests to discuss code changes before merging them into main branch
- Protect branches by branch policies

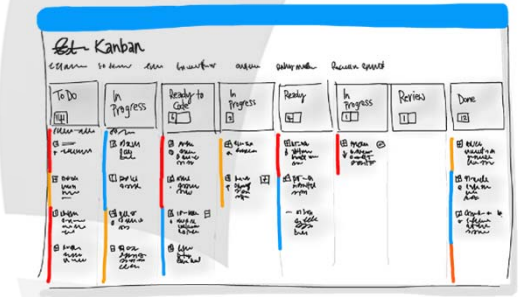
Devops elements



- Agile Planning

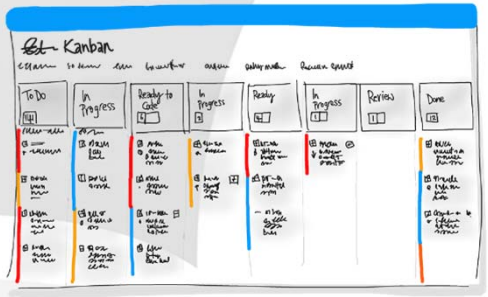


Devops elements – agile



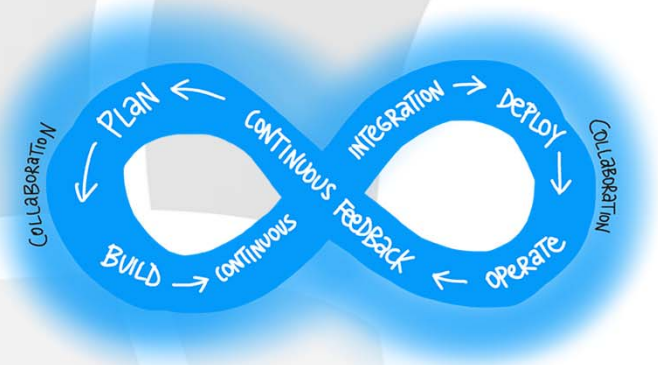
- Agile planning is used to plan and isolate work into sprints, manage team capacity, and help teams quickly adapt to changing business needs

Agile values:

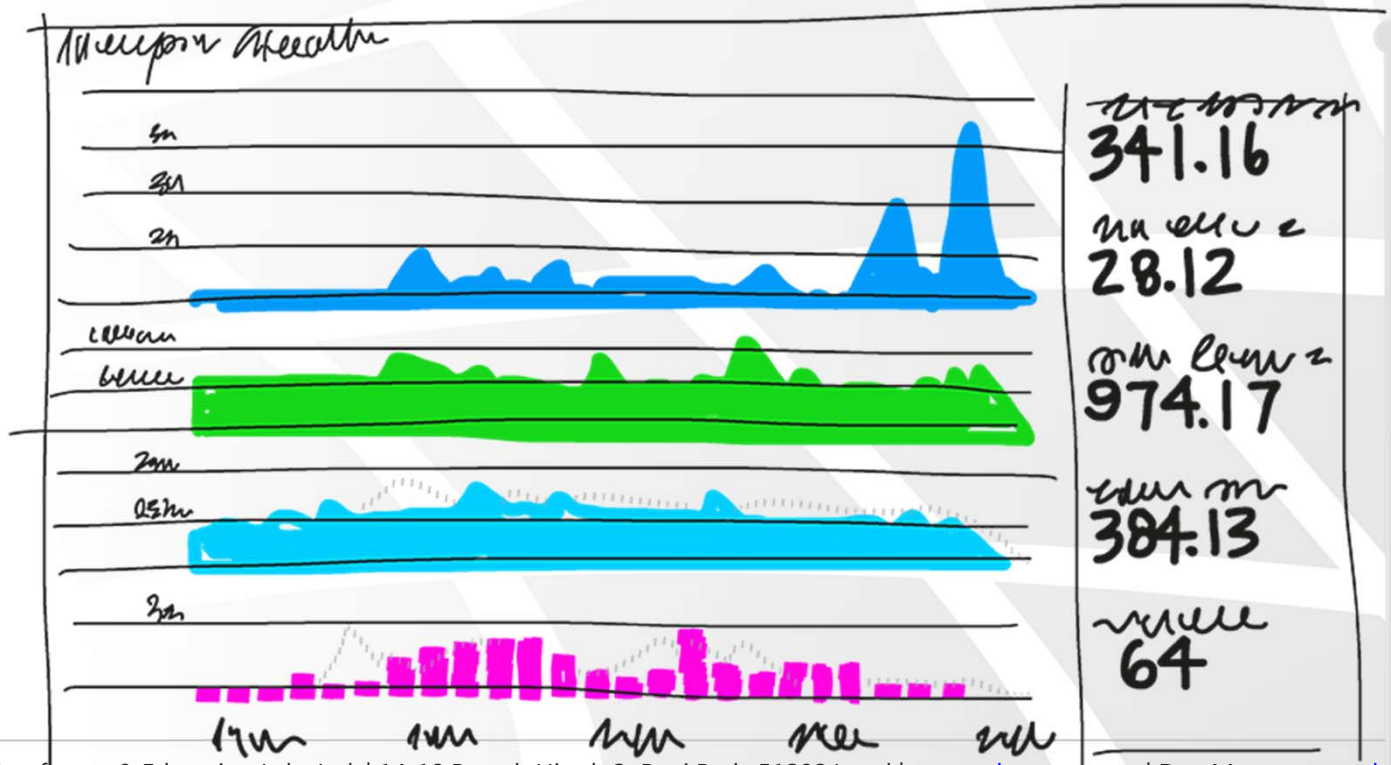


- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

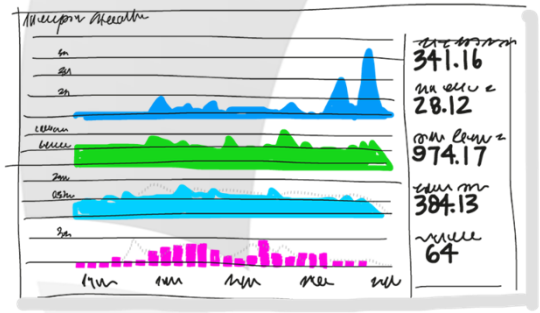
Devops elements



- Monitoring

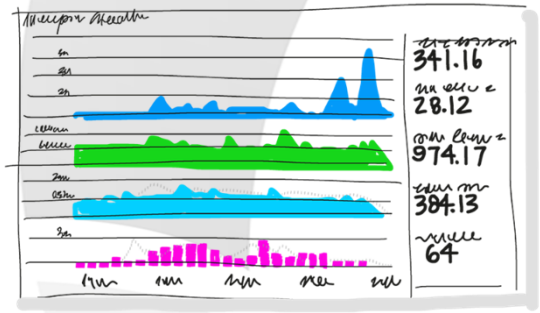


Devops – monitoring



- Monitoring and Logging of running applications including production environments
- for application health / customer usage
- helps quickly validate or disprove strategies.
- Rich data is captured and stored in various logging formats.

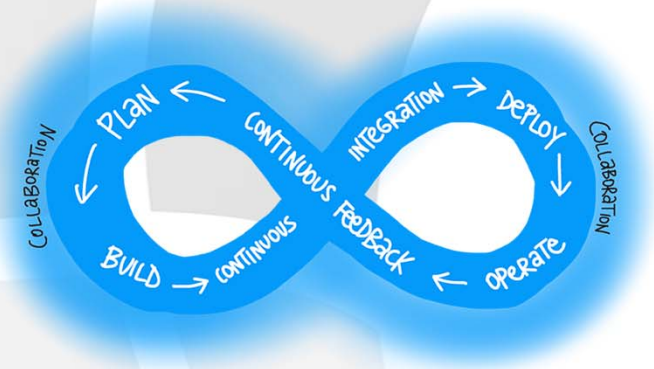
Devops – monitoring



- Monitoring delivers information about an application's performance and usage patterns.
- Monitoring is often used to "test in production".
- Effective monitoring is essential to allow DevOps teams to deliver at speed, get feedback from production, and increase customers satisfaction, acquisition and retention

Devops elements

- Containers



Devops – containers

- operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes
- easily package an application's code, configurations, and dependencies into easy to use building blocks

Devops – containers

- Containers encapsulate all the necessary application files and software dependencies and serve as a building block that can be deployed on any compute resource regardless of software, operating system, or hardware configurations

- Containers are version controlled

Sela devops course

- Day 1 (today) – devops intro + git
- Day 2 – CI (Jenkins + artifactory)
- Day 3 – CD (ansible) + monitoring (graphite & Grafana)
- Day 4 – Docker containers
- Day 5 – Final Workshop

Questions

Dan Morgenstern

Get started with Git

danm@sela.co.il

Agenda

- Module 01: Introduction
- Module 02: Git Structure
- Module 03: Working Locally (basics)
- Module 04: Working Locally (branches)
- Module 05: Working Locally (merge & rebase)
- Module 06: Working Locally (undoing changes)
- Module 07: Working Locally (the stash)

Agenda

- Module 08: Working with Remotes
- Module 09: Git Workflows
- Module 10: What Next?

Module 01: Introduction

Get started with Git

Agenda

- What is Git?
- Version Control Systems (VCS)
- Centralized VS Distributed
- Git – Distributed but Centralized
- Git Server
- Repository Managers
- Git Basics

What is Git?

- The stupid content tracker
- Random 3 letter combination (not used in Unix)
- Stupid, contemptible and despicable (slang)
- Global Information Tracker
- Goddamn Idiotic Truckload of sh*t

distributed version control system

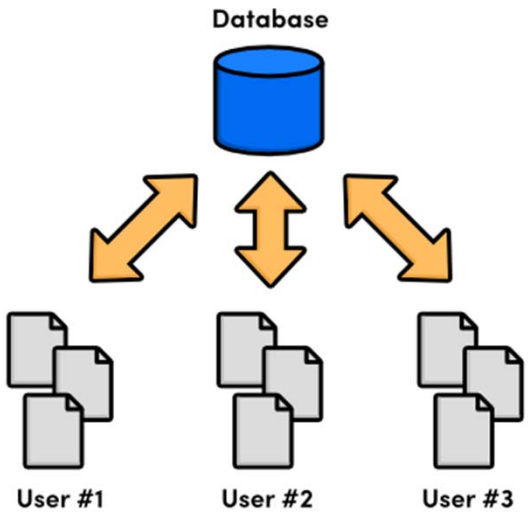
Git is an open source distributed version control system designed with performance, security and flexibility in mind

Version Control Systems (VCS)

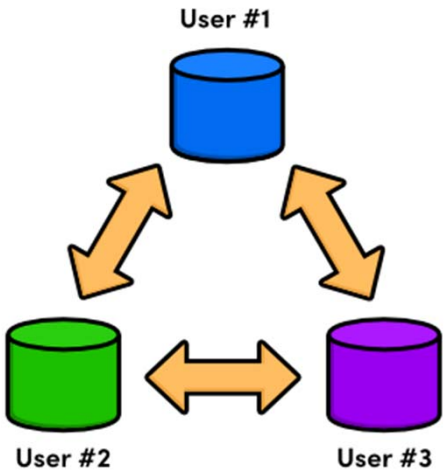
- In general, is a kind of “database” which record changes to a file o set of files over the time.
  - Teamwork
  - Store Versions Properly
  - Show differences between versions
  - Restore previous versions
  - Understand project history
  - Backup

Centralized VS Distributed

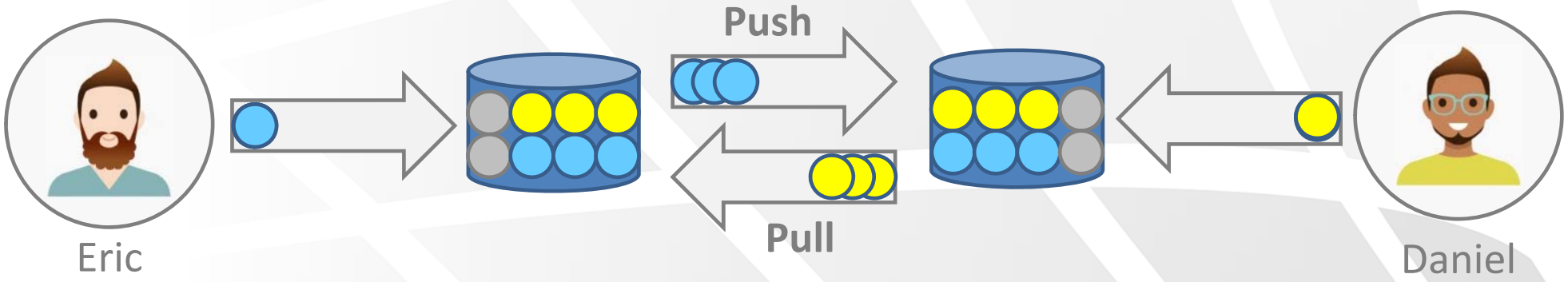
Centralized



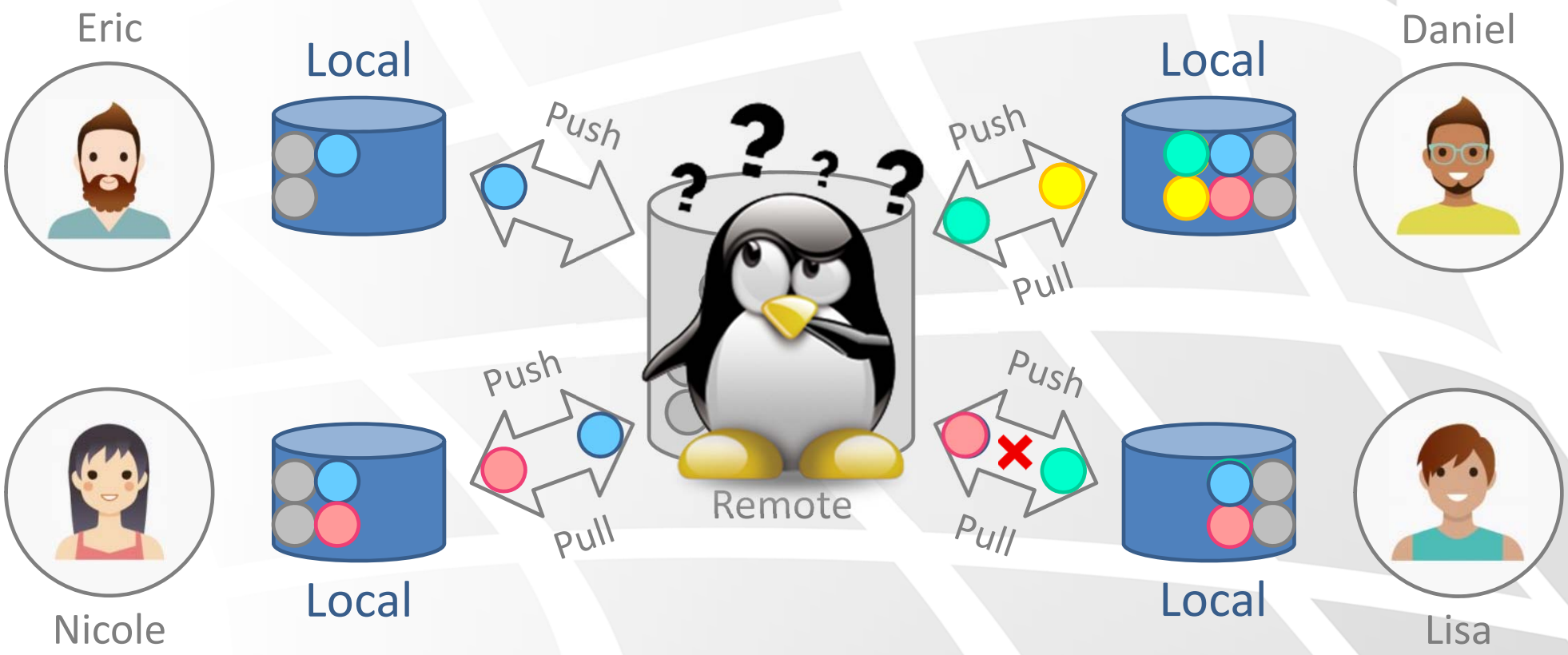
Distributed



Distributed VCS - Explained

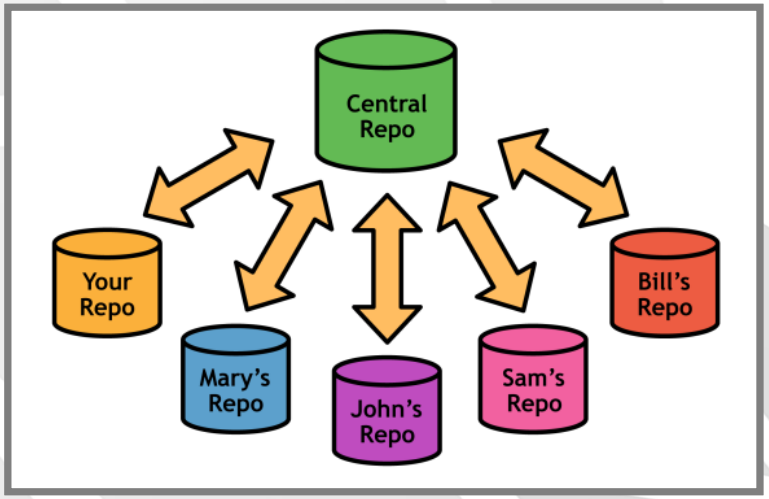


Git – Distributed but Centralized



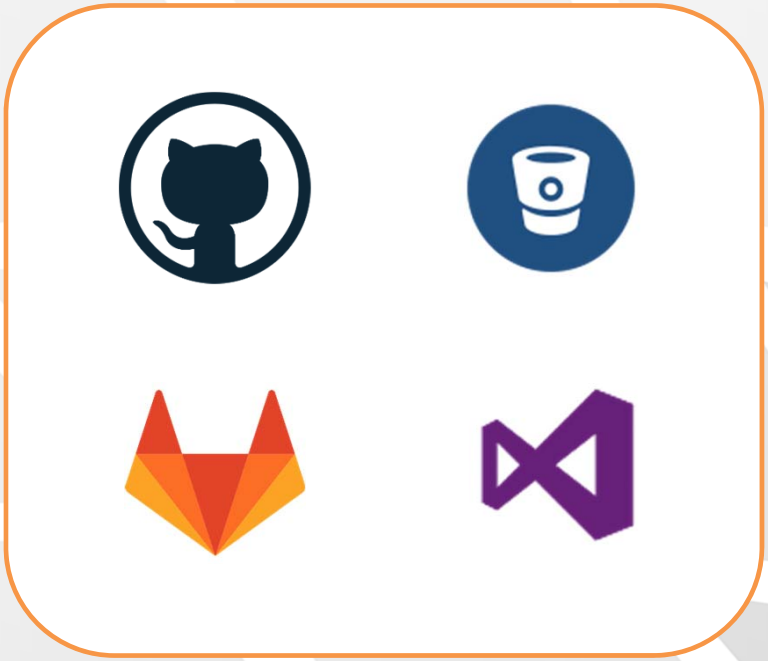
Git Server

- A Git server is just a machine that has Git installed that you and your team can push and pull changes from a Git repository.
- One Git Server can store several repositories.
- Central repositories are often "bare" (no working directory)



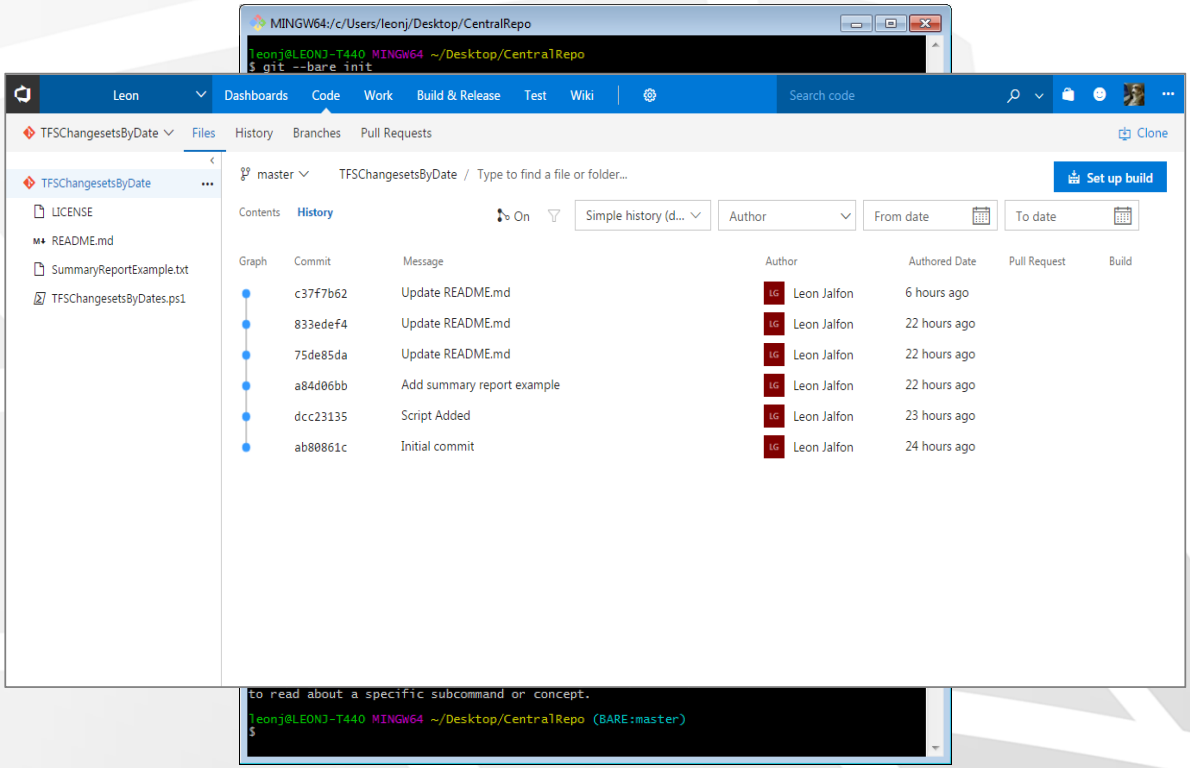
Repository Managers

- Instead of setting up your own server, you can also use a hosting service such as:
  - GitHub
  - GitLab
  - Bitbucket
  - Git-TFS
  - Perforce



Repository Managers

- Manage Security
- Manage Backups
- High Availability
- Manage Repositories
- Groups and Teams
- UI Management Tools
- Issue Tracking
- Code Review Process
- Integrations



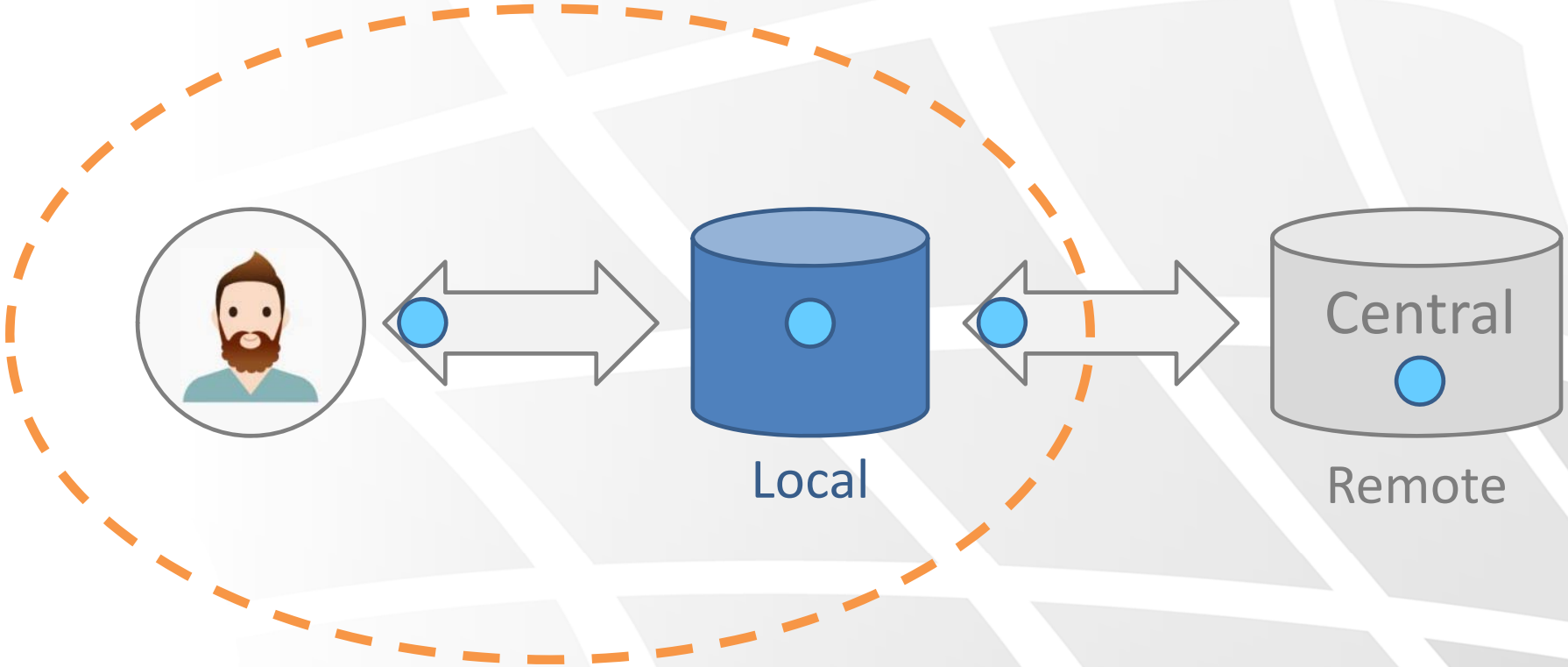
Git Basics

- Git stores snapshots instead of deltas
- Each developer has a copy of the entire repository
- You can continue your work while been offline
- Branches are part of everyday development process
- Merging is central to Git (don’t be afraid of conflicts)
- Git is based on the key-value model

Introduction Summary

Git is a free and open source distributed version control system designed with performance, security and flexibility in mind

Summary



Questions

Module 02: Git Structure

Get started with Git

Agenda

- Meeting the SHA1
- Git Objects
- Git History
- Lab 1: Let's make history…

Git Structure - Meeting the SHA1

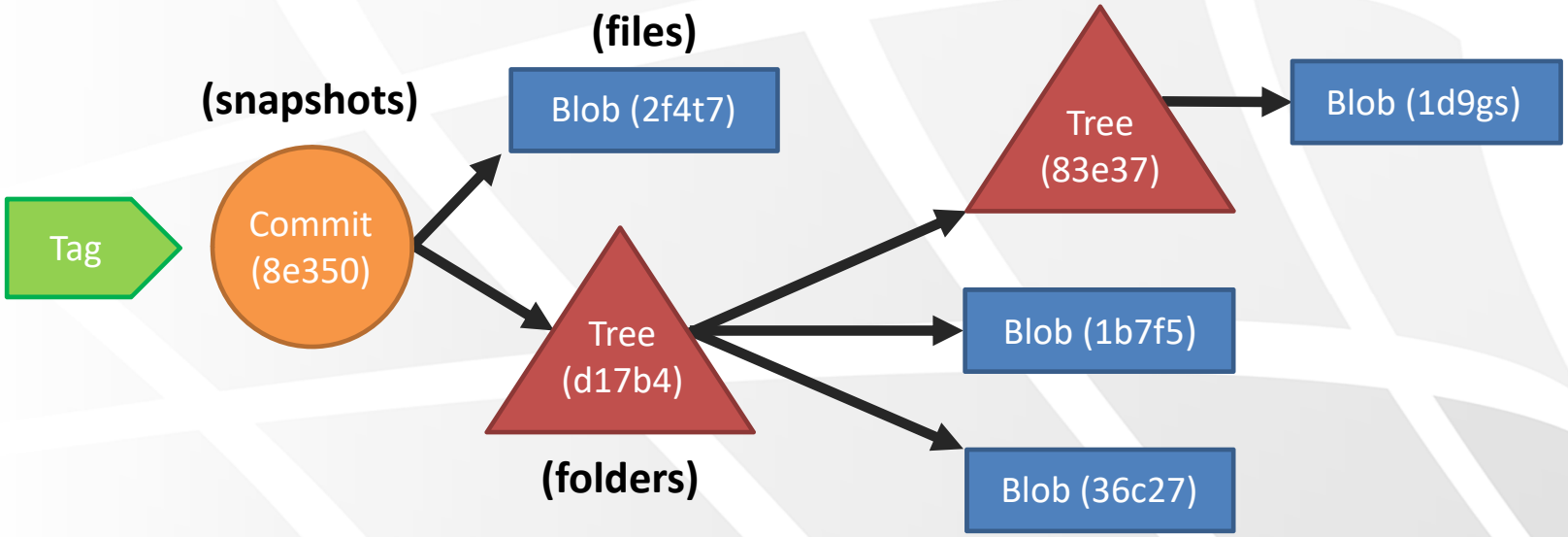
- Is a hash function that convert an long string of data into a 40 character hexadecimal number

SHA1 = **e8964**2b96685d5f22ee7044e05b9e6566e69b7a5

- Every object in Git have its own SHA1 (used as key)
- Each SHA1 is unique (or almost)
- Usually only the first 5 digits are used

$$\frac{1}{1.050.000.000}$$

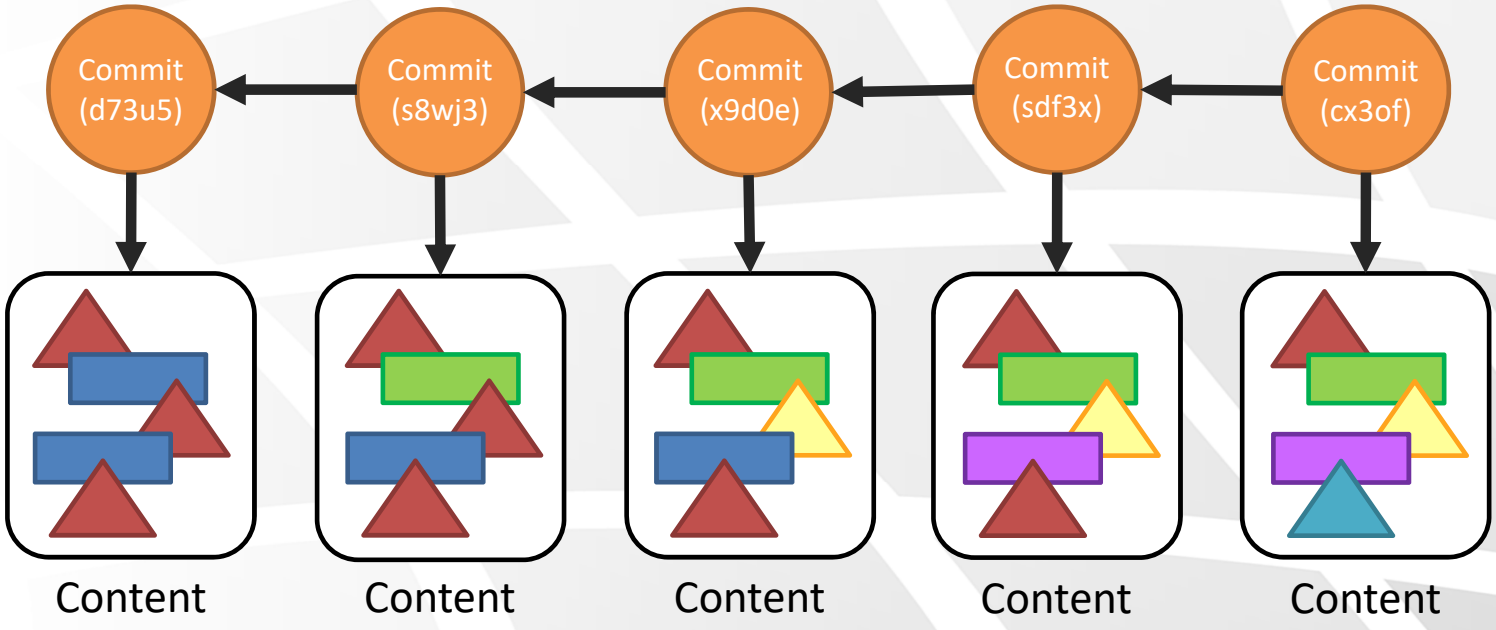
Git Structure - Objects



- A commit is a snapshot at some point in time
- A tag is a reference to a commit

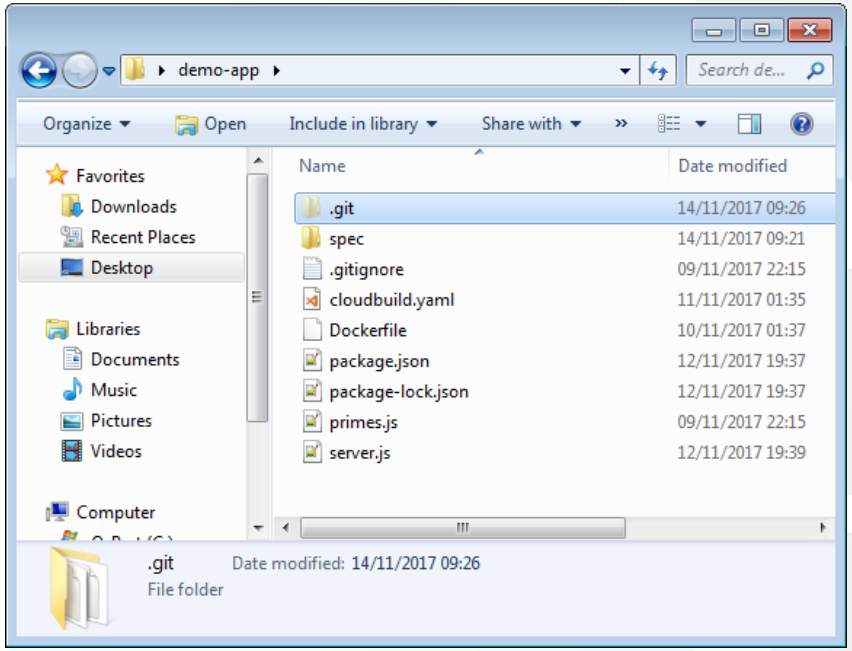
Git Structure - History

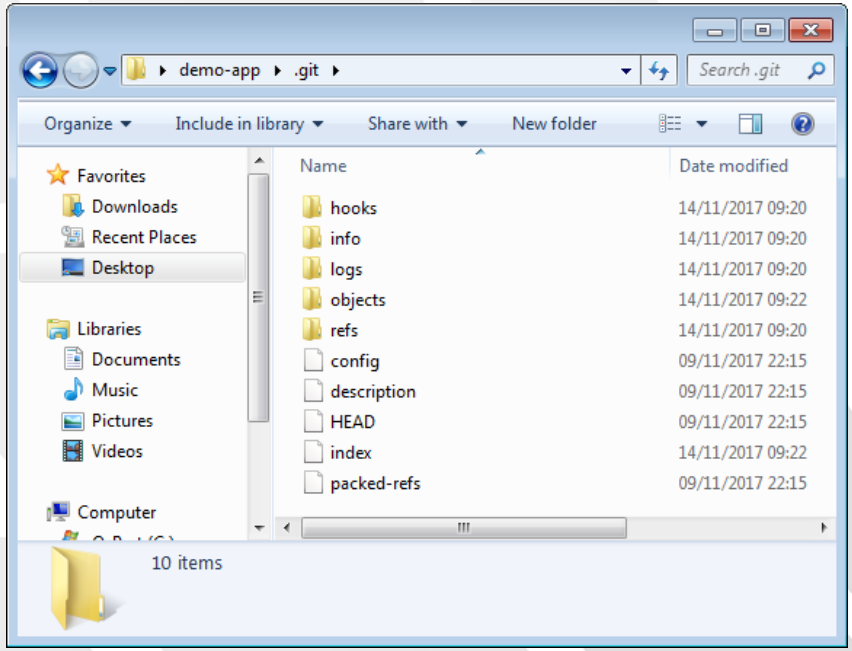
- The history is a set of interconnected commits



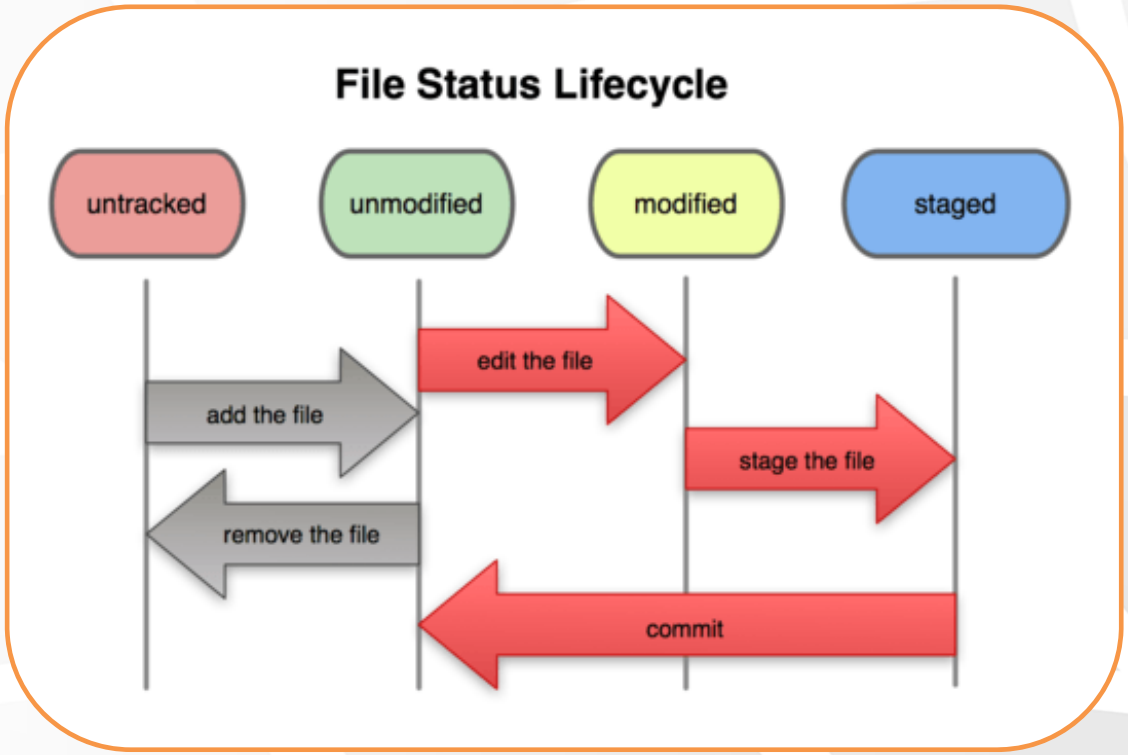
Git Structure – How things are stored

- The whole repository is stored under the `.git` folder





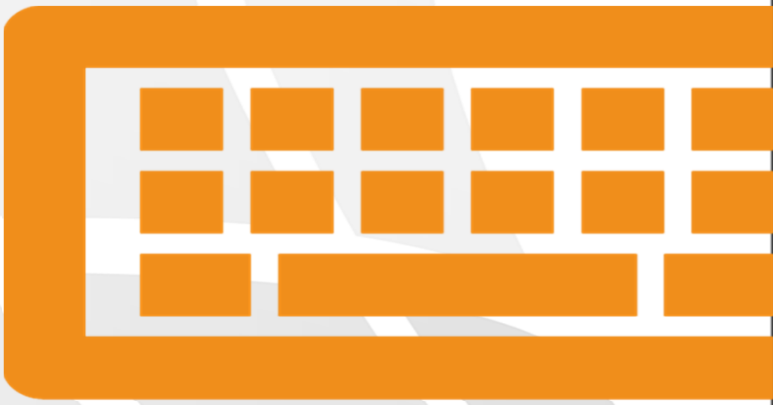
Git Structure – Files Status Lifecycle



Questions

Lab 1: Let's make history…

Lab



https://gitlab.com/git-getstarted/lab1

Module 03: Working Locally (basics)

Get started with Git

Agenda

- The Four Areas
- Learning Commands
- Basic Commands:
  - $ git init
  - $ git clone
  - $ git add
  - $ git commit
  - $ git rm
  - $ git mv
- Lab 2: Basic Commands

The Four Areas

Stash

Working Area

Index

Repository

Working Locally (Commands)

- Knowing a command means understanding how it affects the 4 areas
- The best way to learn the commands is by using the command line
- Git clients are good as long as you understand what they are doing

Working Locally (`git init`)


```
$ git init

Initialized empty Git repository in
C:/Users/leonj/Desktop/MyRepo/.git/
```

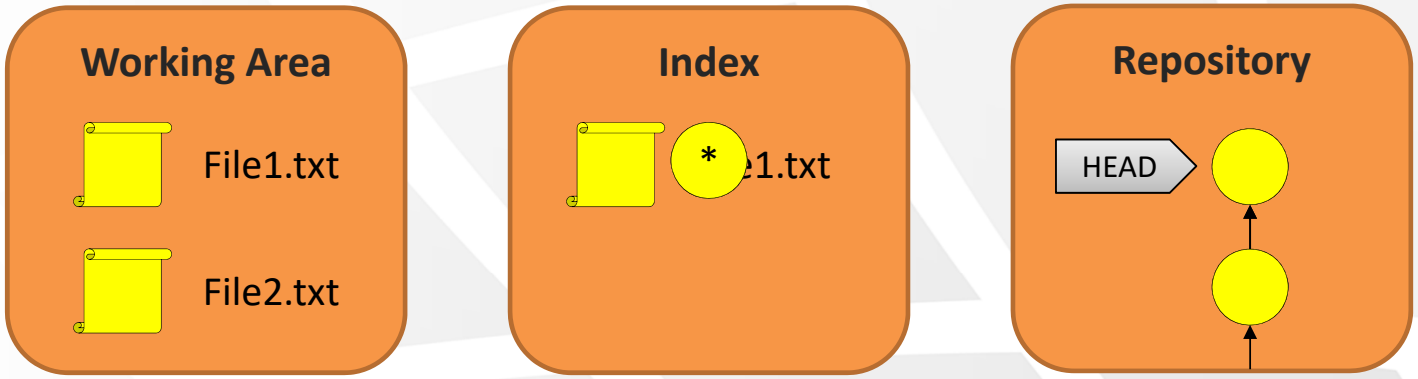
Working Locally (git clone)


```
$ git clone <RepoUrl> “Folder”
  Cloning into ‘Folder’...
  Unpacking objects: 100% (19/19), done.
```

Working Locally (`git add`)

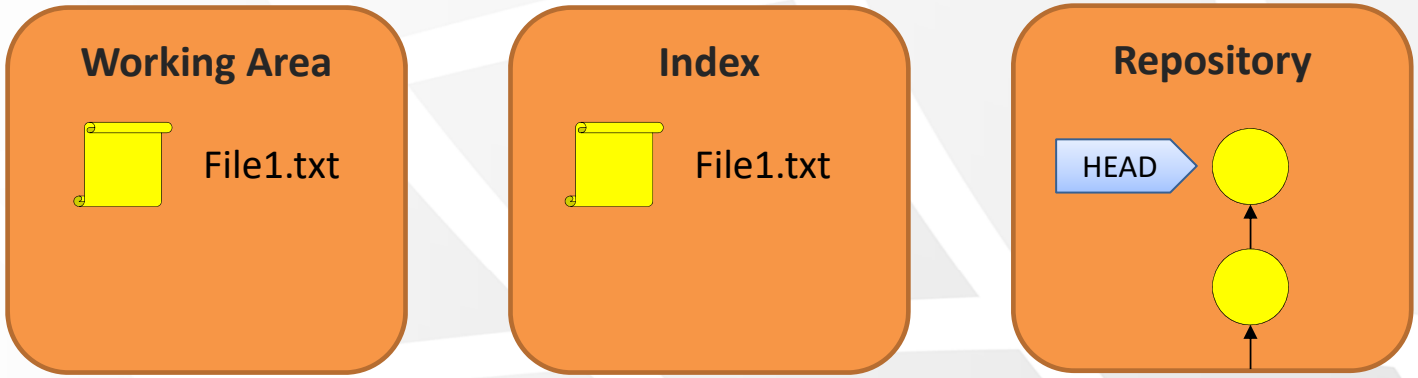

```
$ git add "File1.txt"
$ git status
  Changes to be committed:
        modified:   File1.txt
```

Working Locally (git commit)



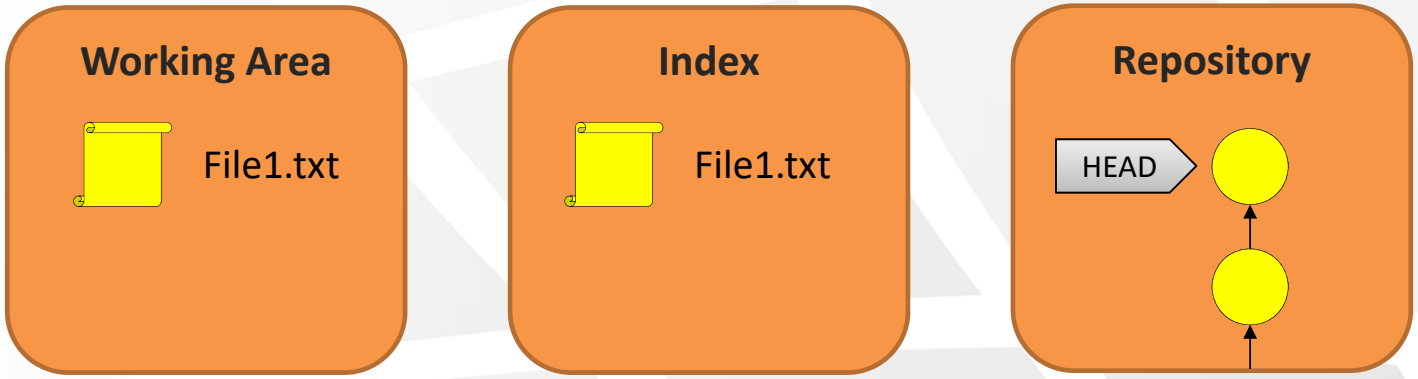

```
$ git commit –m “message”

  [master (root-commit) 22f1a52] message
   1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 MyFile.txt
```

Working Locally (git rm --cached)



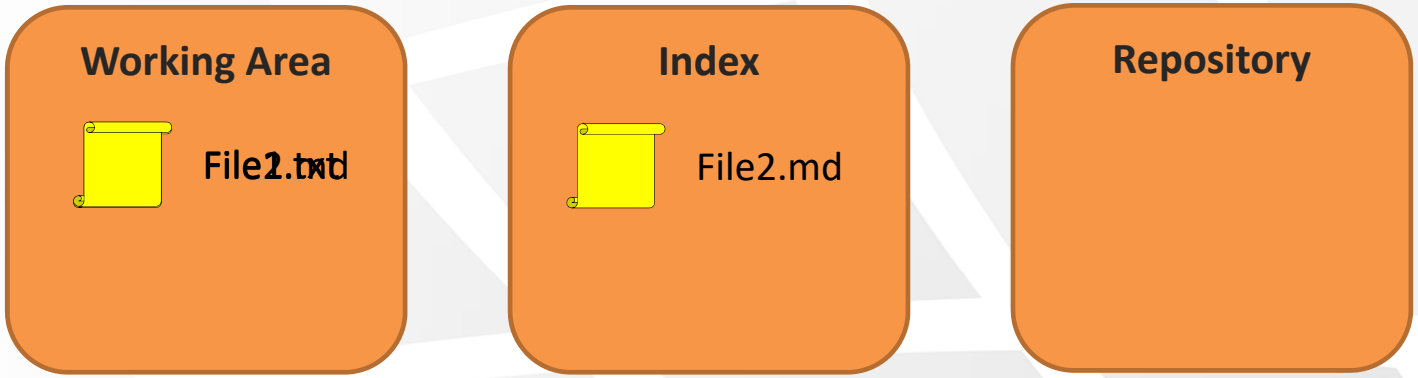

```
$ git rm "File1.txt"
  error: the following file has changes staged in the index
  (use --cached to keep the file, or -f to force removal)
$ git rm --cached "File1.txt"
```

Working Locally (git rm -f)




```
$ git rm "File1.txt"
  error: the following file has changes staged in the index
  (use --cached to keep the file, or -f to force removal)
$ git rm -f "File1.txt"
```

Working Locally (git mv)

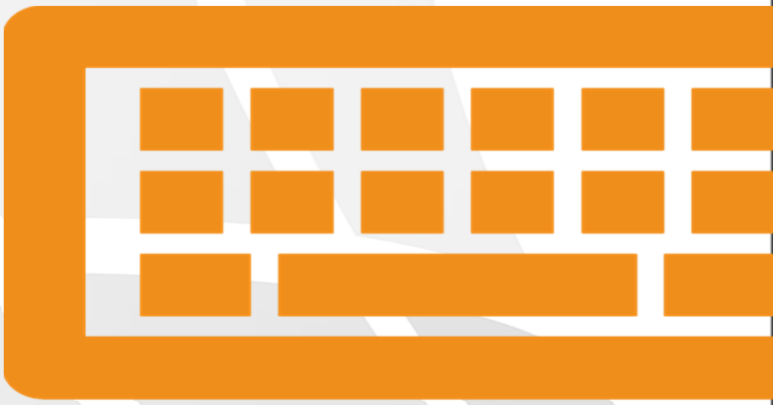



```
$ git mv “File1.txt” “File2.md”
$ git status
  Changes to be committed:
        renamed:   File1.txt-> File2.md
```

Questions

Lab 2: Basic Commands

Lab



https://gitlab.com/git-getstarted/lab2

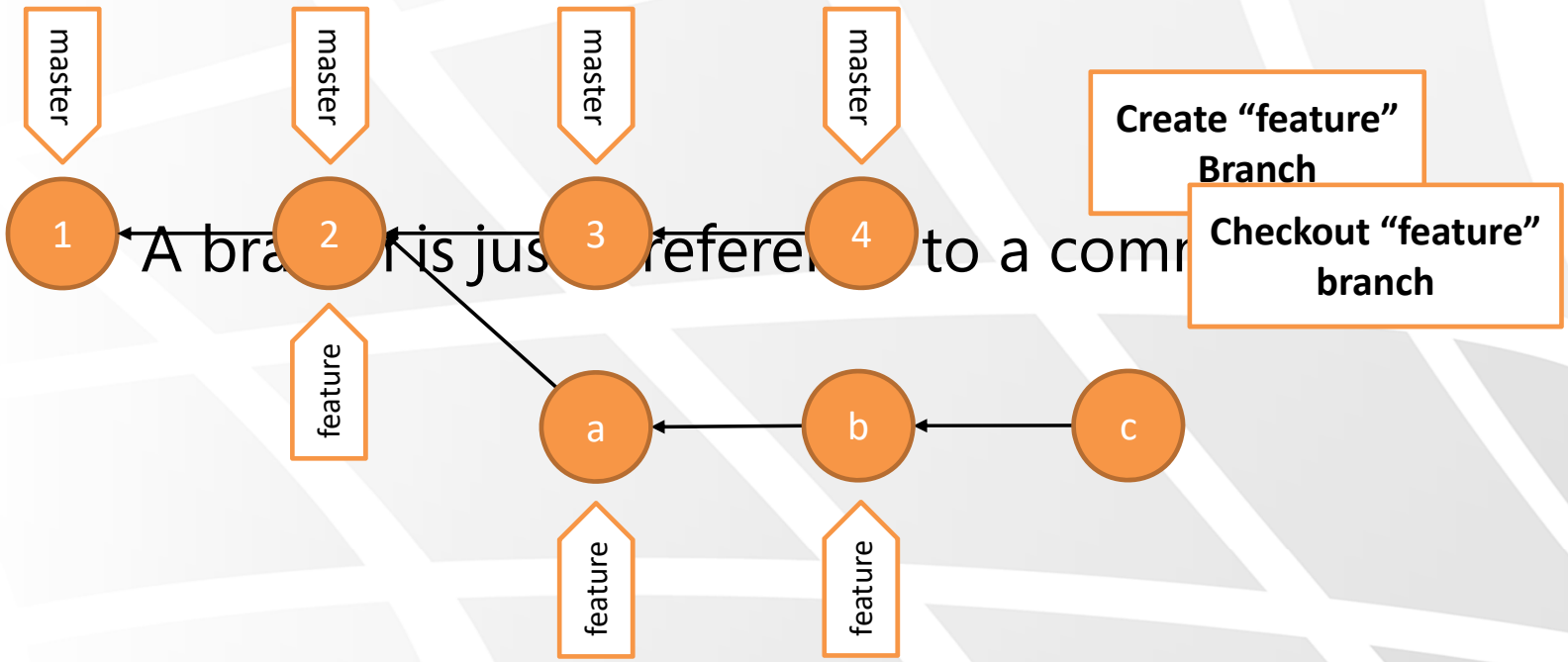
Module 04: Working Locally (branches)

Get started with Git

Agenda

- What Branches really are?
- Loosing the Head
- Relative References
- Commands:
  - $ git checkout
  - $ git branch
- Lab 3: Moving between branches and commits

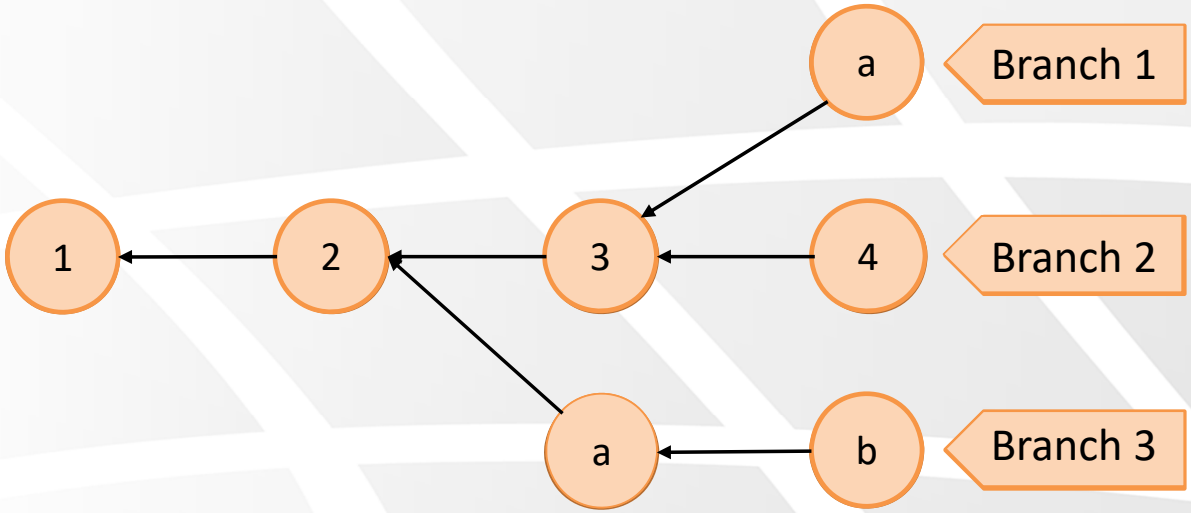
Working Locally – What Branches really are?



A branch is just a reference to a commit

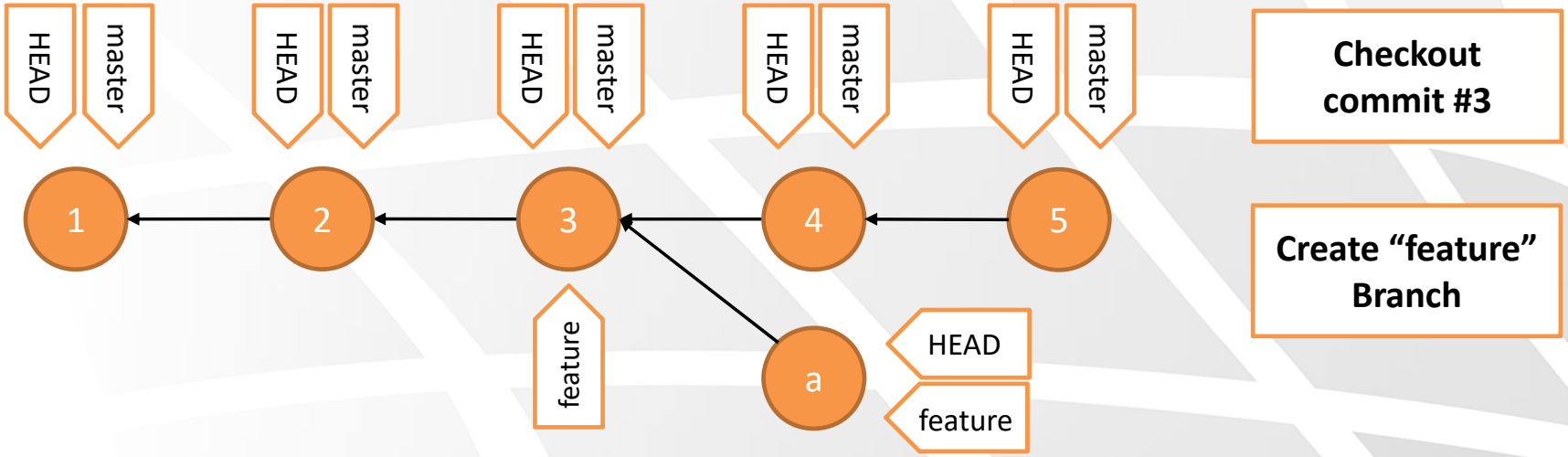
Working Locally – What Branches really are?

Or rather, the entrance to the sequence…



Working Locally – Loosing the Head

- HEAD is just a reference to the current commit



Working Locally – Loosing the Head

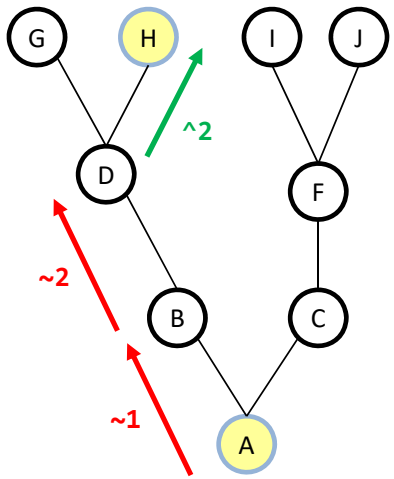
- HEAD is a reference to the current commit
- Happens when the HEAD is not referencing the branch tip
- It is risky to create commits directly from a detached head
- To create a new commit you'd better create a new branch

Working Locally – Relative References

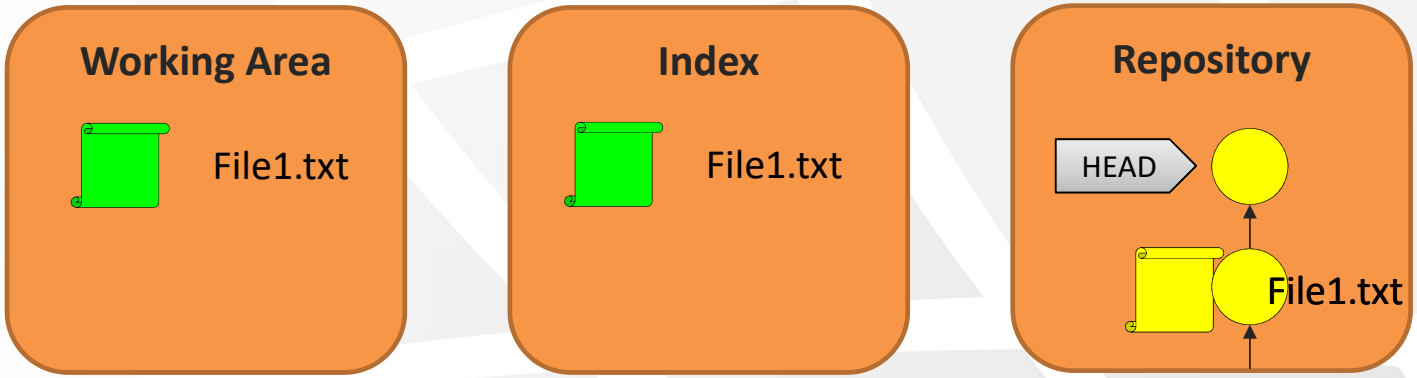
Using x~n
(n commits before x)

Using x^n
(n parent of x)

H	=	A	~2	^2

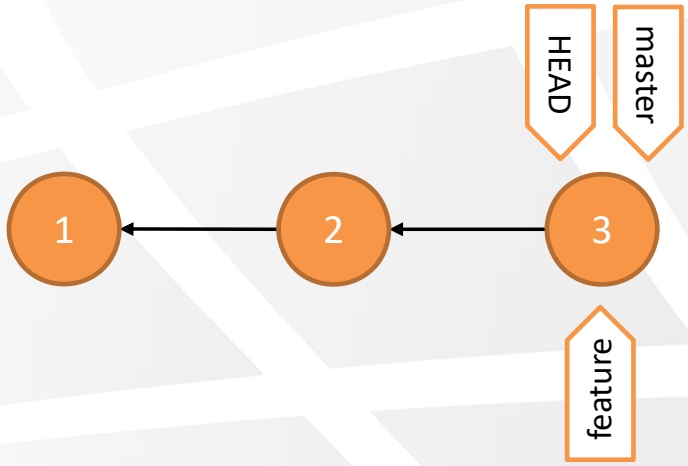


Working Locally (git checkout)



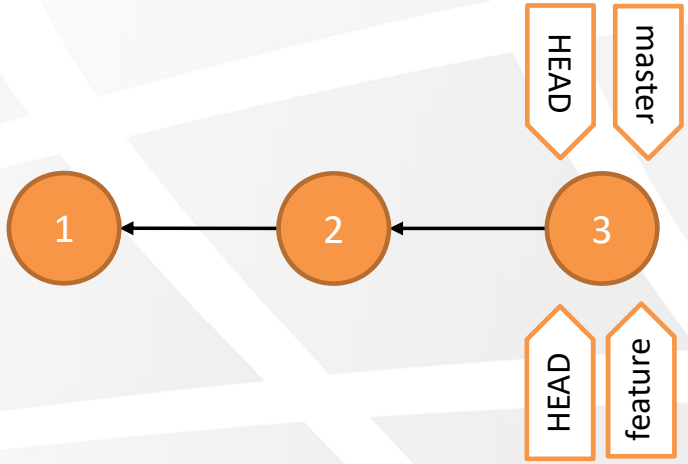

```
$ git checkout HEAD~1

  Switched to branch ‘master’

  Your branch is up-to-date with 'origin/master'
```

Working Locally (git branch)



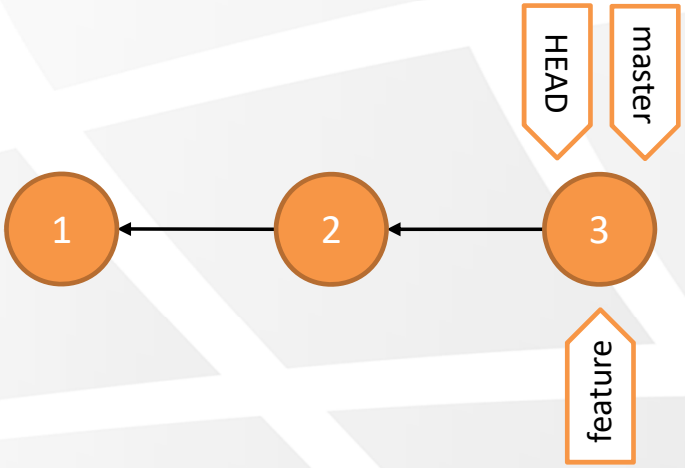

```
$ git branch “feature”

$ git branch
  feature
* master
```

Working Locally (`git checkout -b`)




```
$ git checkout -b "feature"
   Switched to a new branch 'feature'
$ git branch
 * feature
   master
```

Working Locally (git branch -d)

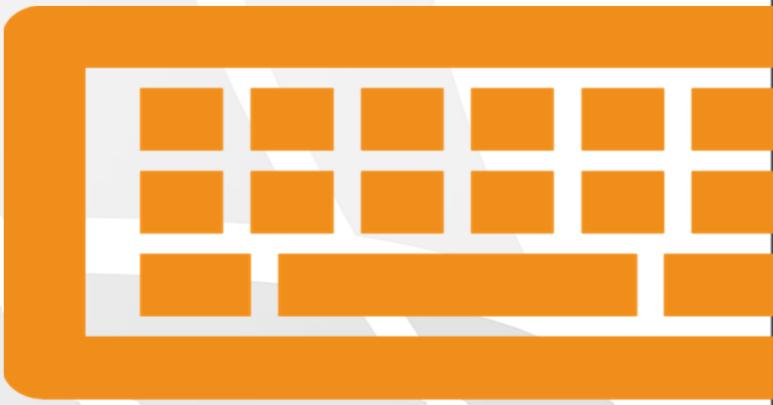



```
$ git branch -d “feature”
  Deleted branch feature (was 41f1e7a).
```

Questions

Lab 3: Moving Between Branches and Commits

Lab



https://gitlab.com/git-getstarted/lab3

Module 05: Working Locally (Merge & Rebase)

Get started with Git

Agenda

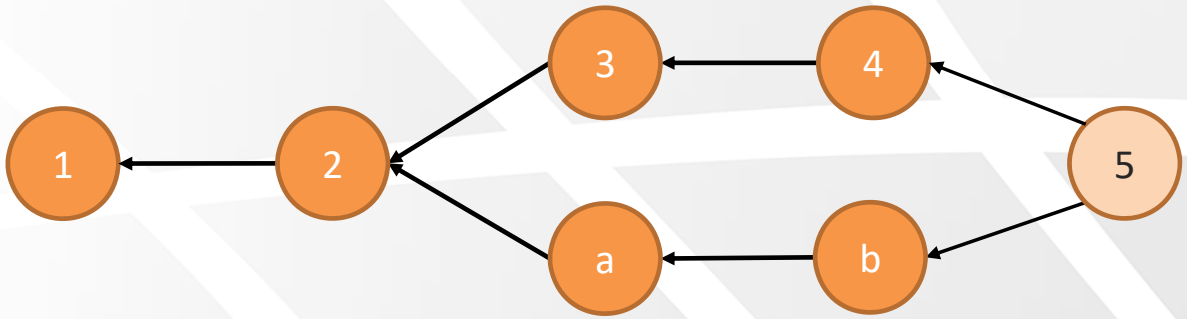
- What is Merge?
- What is Rebase?
- Resolving Conflicts
- Merging VS Rebasing
- Commands:
  - $ git merge
  - $ git rebase
- Lab 4: Merging and Rebasing

Working Locally - Merging & Rebasing

- What is Merge?
- What is Rebase?
- When to Merge and when to Rebase?

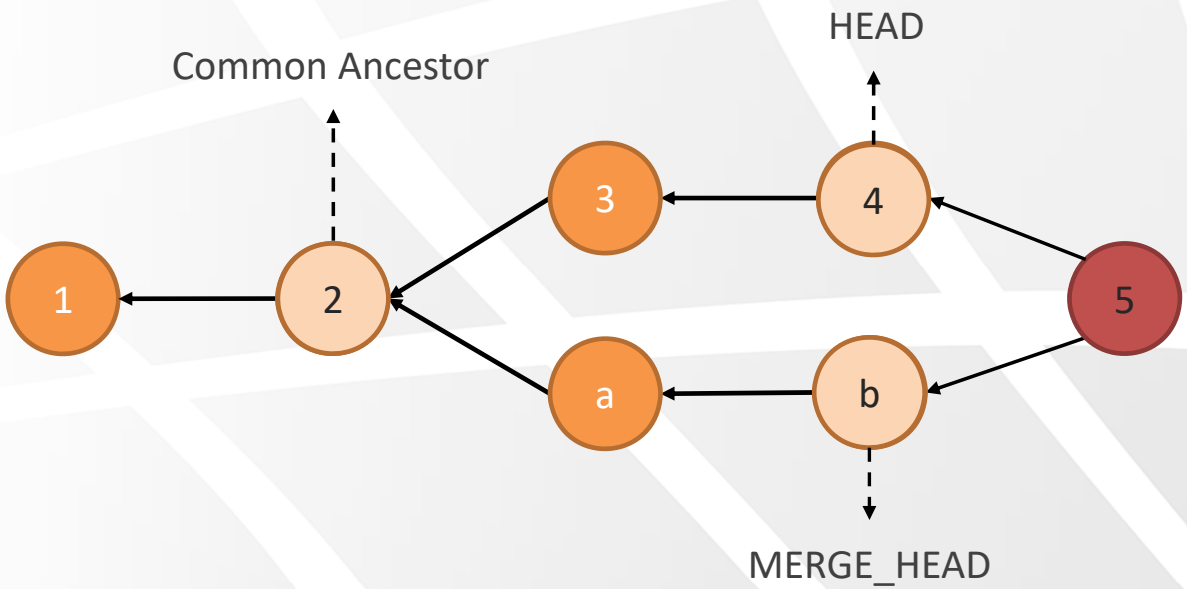
Working Locally - Merging Simplified

- Merge is just a commit with two parents



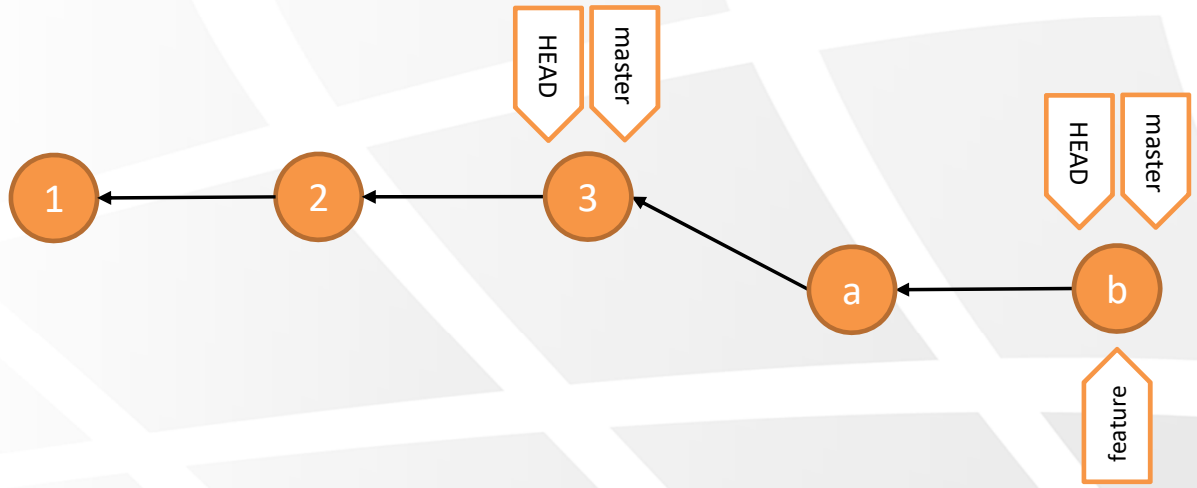
- Merge command says: “merge the branch X into the current branch”

Working Locally – 3 Way Merge Algorithm



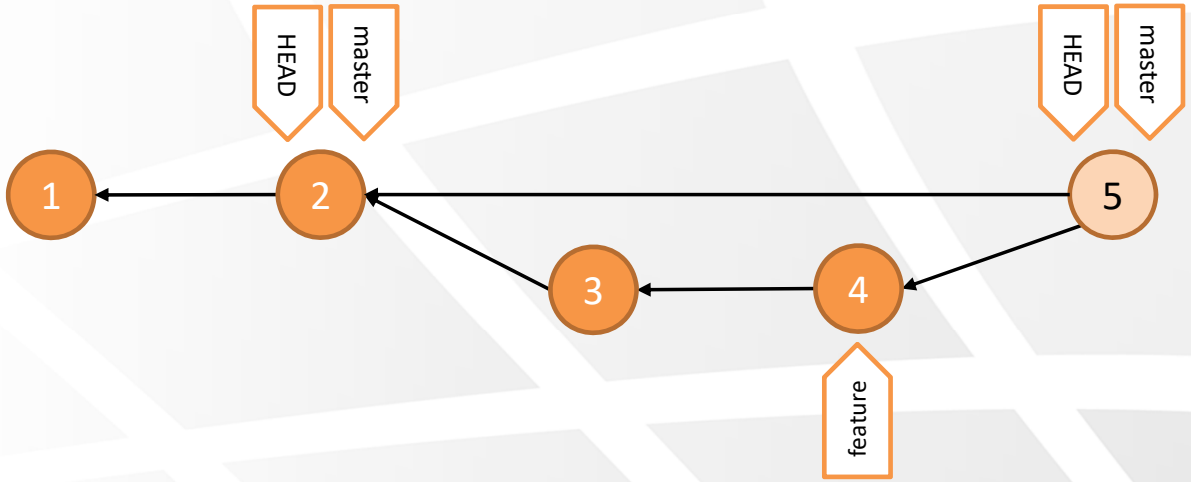
- Compare both commits to merge with it common ancestor

Working Locally (git merge|Fast-Forward)




```
$ git checkout master

$ git merge feature
   Updating b01bdbe..18ddbab
   Fast-forward
```

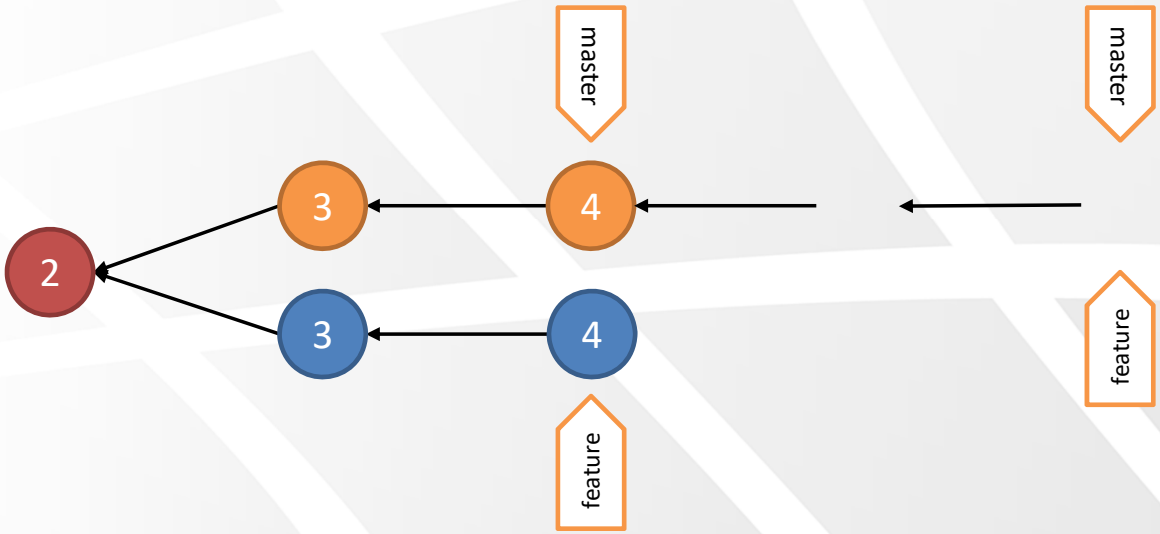
Working Locally (git merge|--no-ff)




```
$ git checkout master
  Switched to branch 'master'

$ git merge feature --no-ff
  Merge made by the 'recursive' strategy.
```

Working Locally – Rebasing Explained

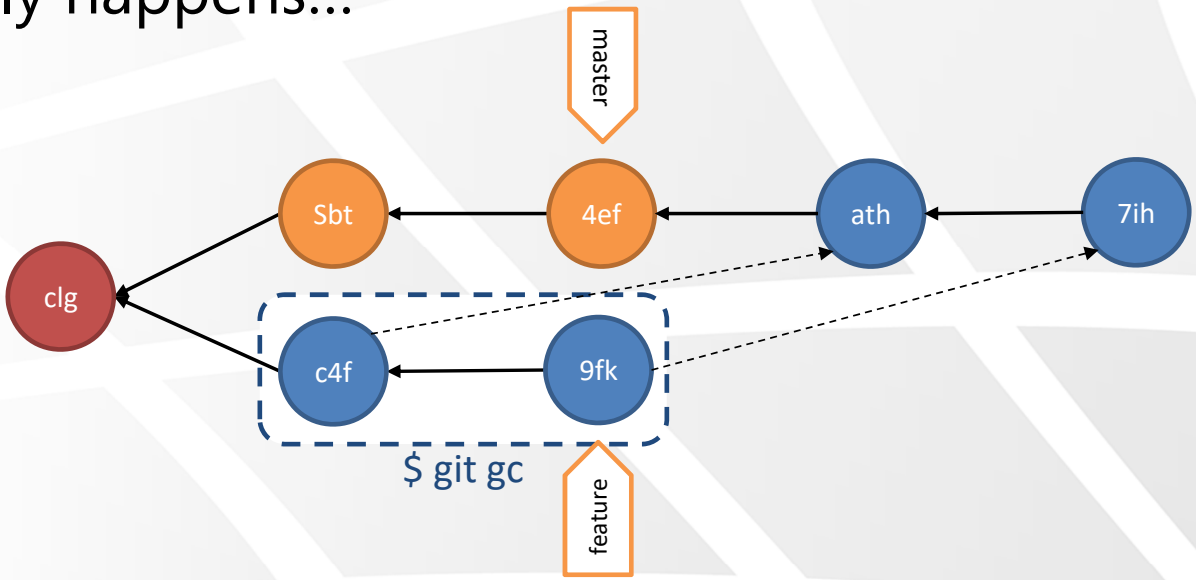
- How it Looks…



- Commits 3 and 4 are moved and references are updated

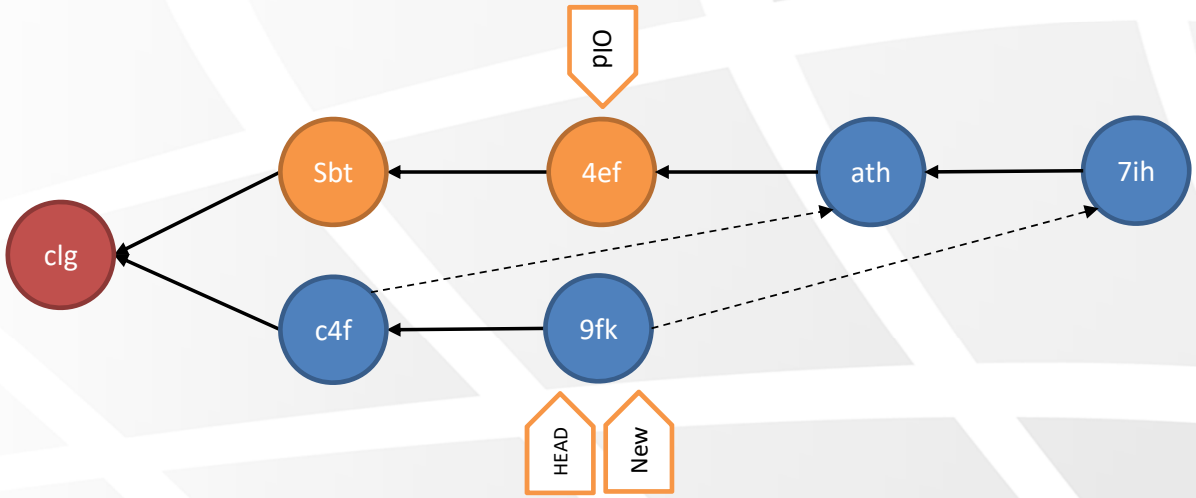
Working Locally – Rebasing Explained

- What really happens…



- Commits 3-4 are deleted and identical commits are created instead

Working Locally (git rebase)




```
$ git checkout New
$ git rebase Old
  First, rewinding head to replay your work on top of it...
  Applying: ...
```

Working Locally - Resolving Conflicts

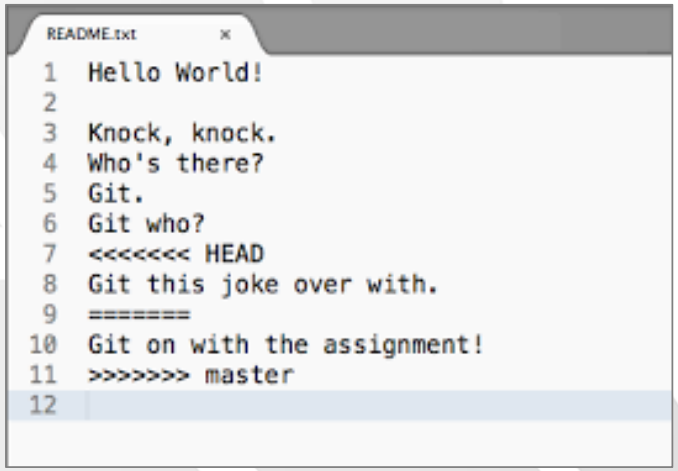
1) Open the file

2) Fix the conflict

3) Commit your changes

- You can optionally use a 'merge tool'
- You can abort the merge/rebase using:

```
$ git merge --abort
$ git rebase --abort
```



(Conflict between the current branch and master)

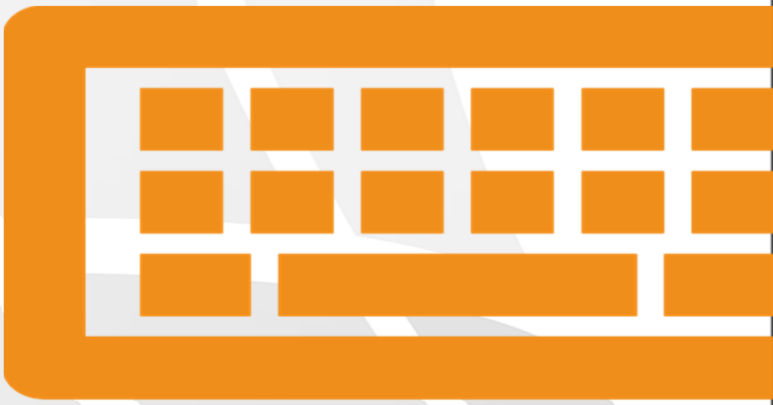
Working Locally - Merging VS Rebasing

- If you aren't sure what you are doing, use merge (rebase can be dangerous)
- Use rebase to keep a clean story (rewrite history)
- Use merge to have a detailed history (although sometimes messy)
- Interactive rebase is great for cleaning changes before pushing
- Do not rebase pushed commits

Questions

Lab 4: Merging and Rebasing

Lab



https://gitlab.com/git-getstarted/lab4

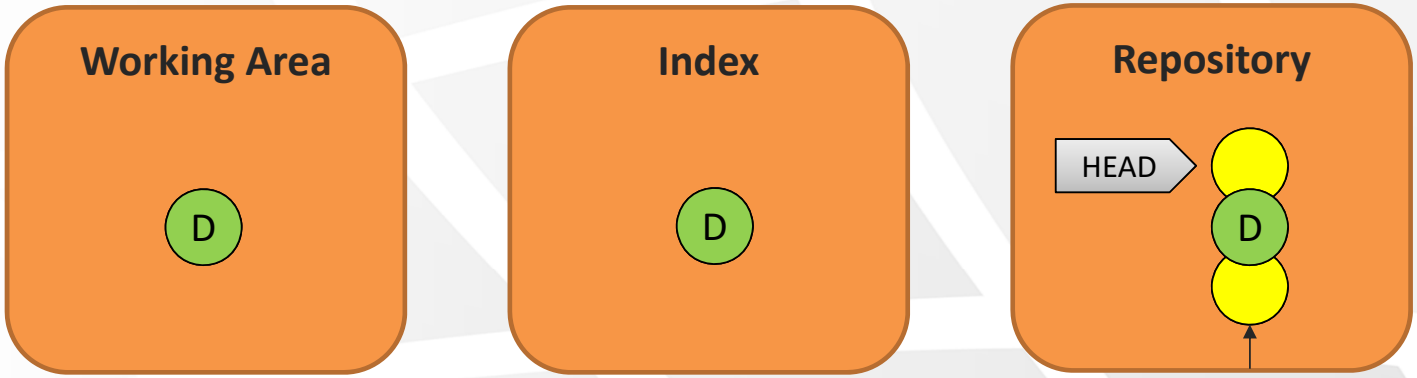
Module 06: Working Locally (Undoing Changes)

Get started with Git

Agenda

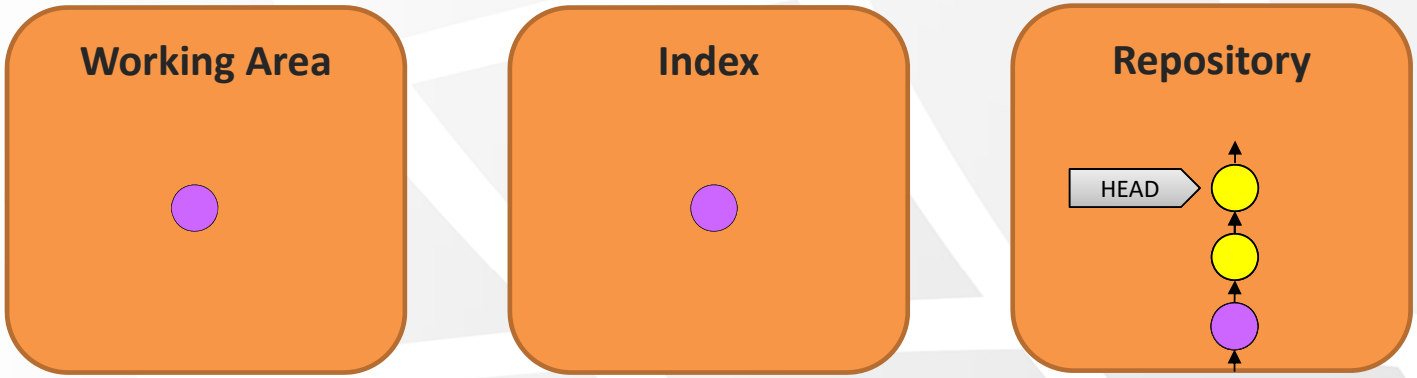
- Commands:
  - $ git revert
  - $ git cherry-pick
  - $ git reset
- Lab 5: Undoing Changes

Working Locally (git revert)




```
$ git revert HEAD

  Revert "NewCommitMessage"

  This reverts commit 6b32e600dff05e6b27e0e42eeb829ee735134336
```

Working Locally (git cherry-pick)



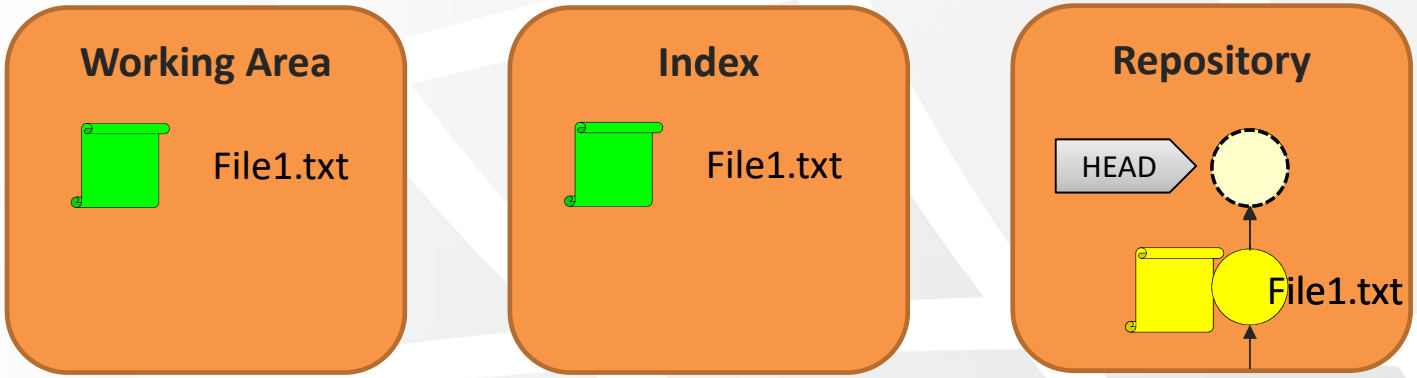

```
$ git cherry-pick HEAD~2

  [master c85d13a] commit
   Date: Sat Mar 11 18:38:24 2017 +0200
   1 file changed, 1 insertion(+)
```

Working Locally `(git reset)`

$git reset moves the current branch, and optionally copies data from the Repository to the other areas

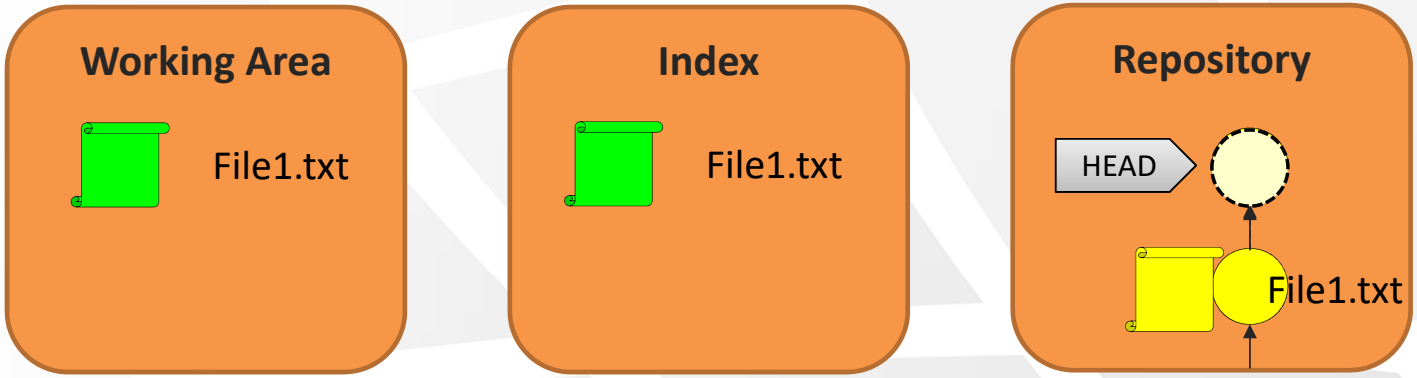
`--soft` `--mixed` `--hard`

Working Locally (`git reset --hard`)



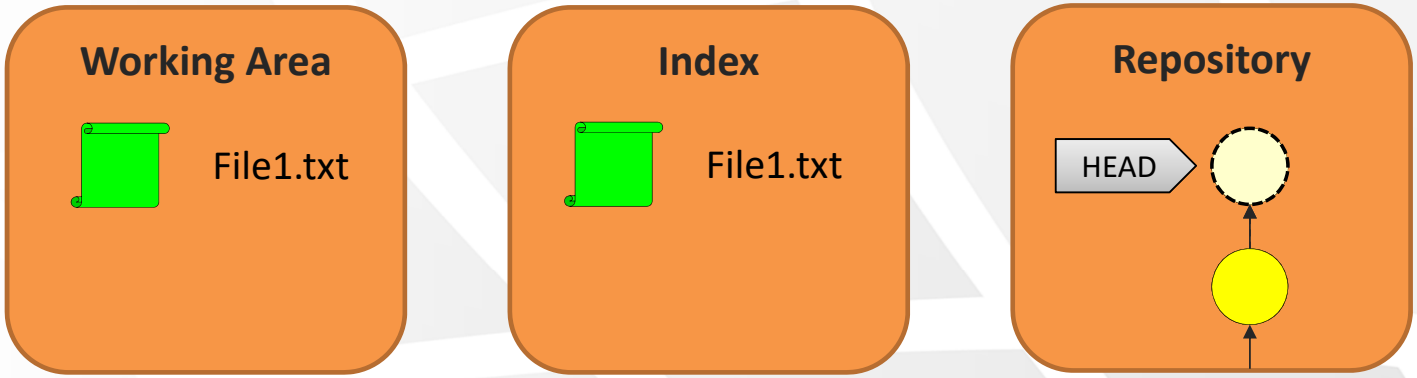

```
$ git reset --hard HEAD~1

  HEAD is now at ec288f2 CommitMessage
```

Working Locally (`git reset --mixed`)




```
$ git reset --mixed HEAD~1

  HEAD is now at ec288f2 CommitMessage
```

Working Locally `(git reset --soft)`

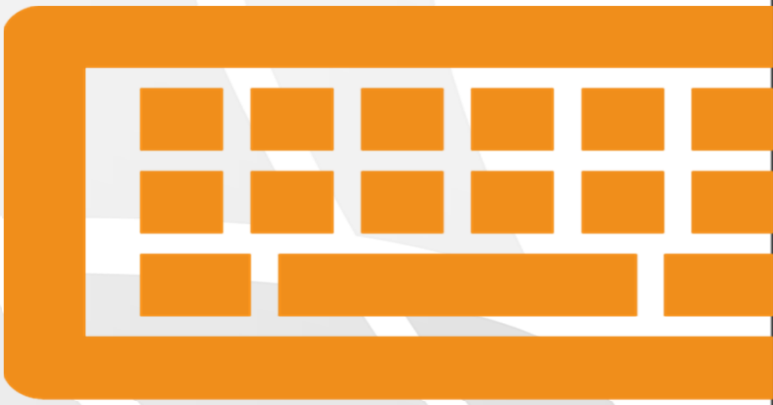



```
$ git reset --soft HEAD~1

  HEAD is now at ec288f2 CommitMessage
```

Questions

Lab 5: Undoing Changes

Lab



https://gitlab.com/git-getstarted/lab5

Module 07: Working Locally (Stashing Changes)

Get started with Git

Agenda

- The Stash Area
- Commands:
  - $ git stash
  - $ git stash list
  - $ git stash apply
  - $ git stash drop
  - $ git stash clear
- Lab 6: Stashing Changes

The Four Areas

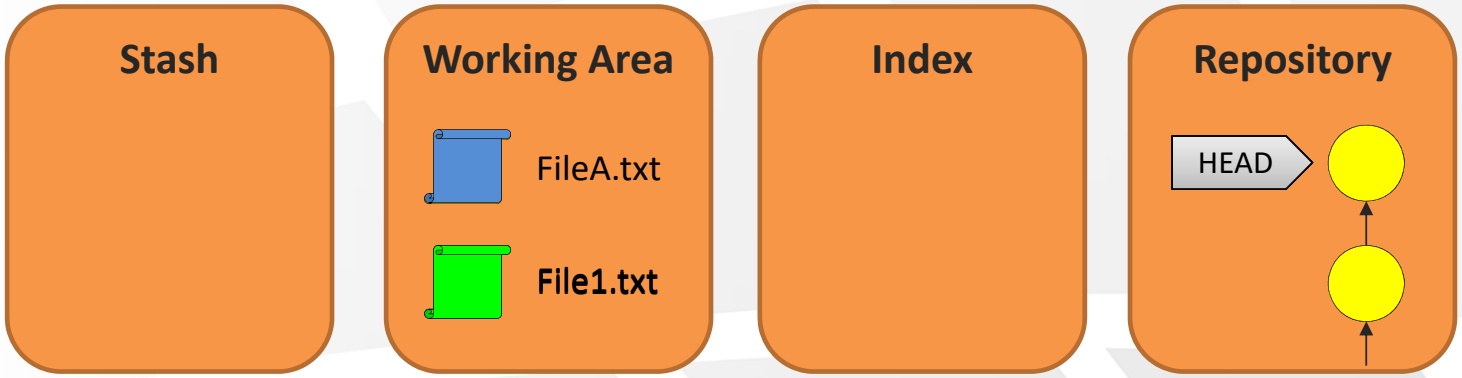
Stash

Working Area

Index

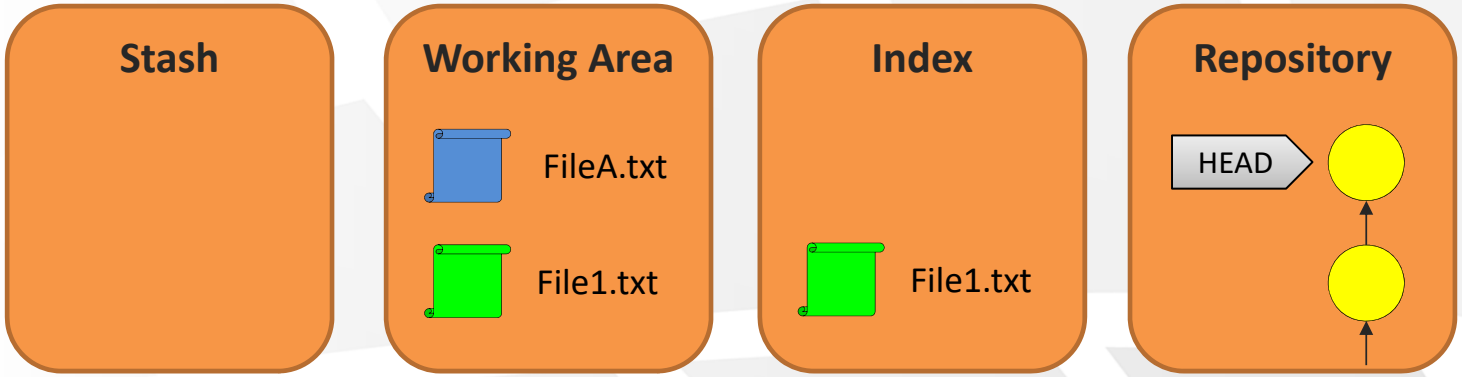
Repository

Working Locally (git stash)



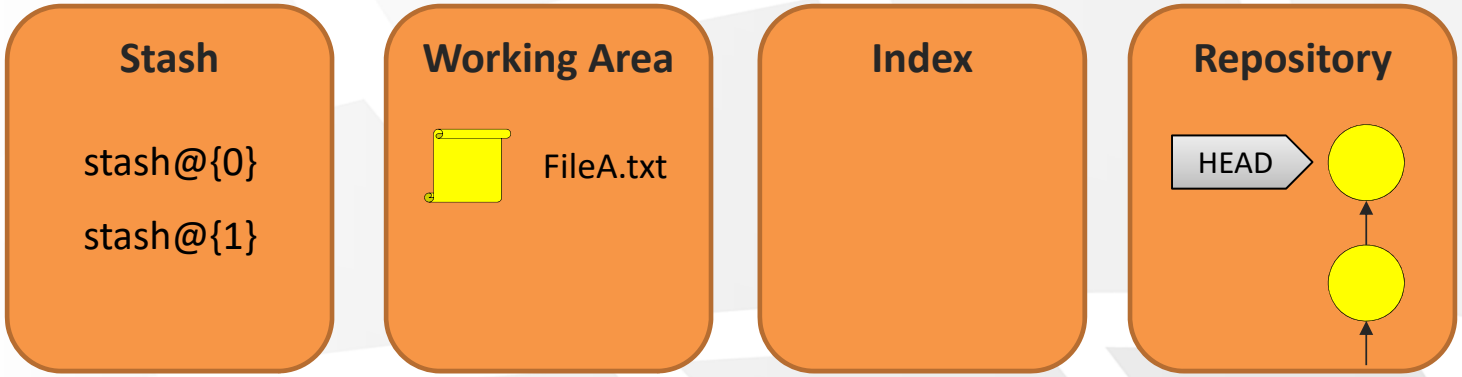

```
$ git stash

  Saved working directory and index state WIP on master:
  c85d13a commit
  HEAD is now at c85d13a commit
```

Working Locally (git stash)



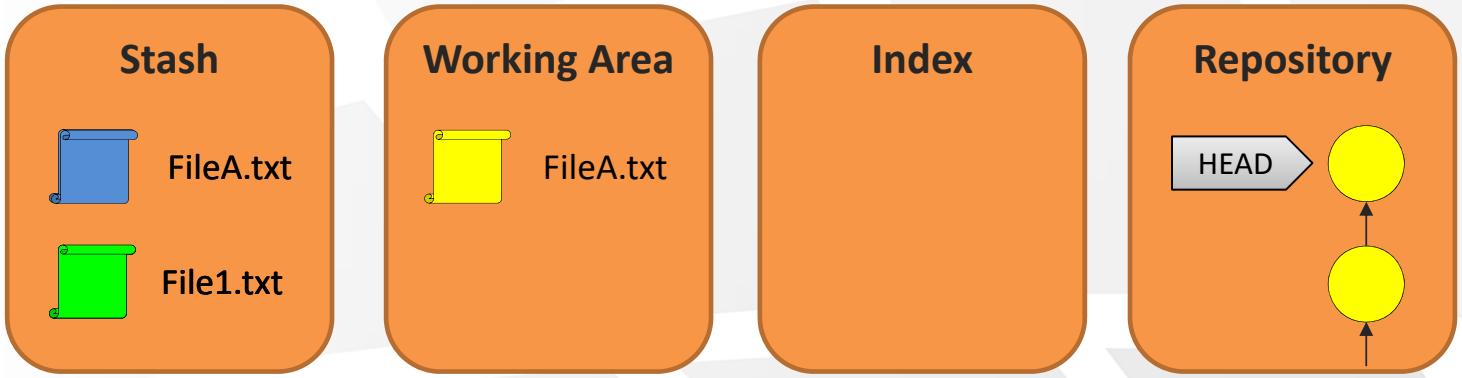

```
$ git stash --include-untracked

  Saved working directory and index state WIP on master:
  c85d13a commit
  HEAD is now at c85d13a commit
```

Working Locally (git stash list)



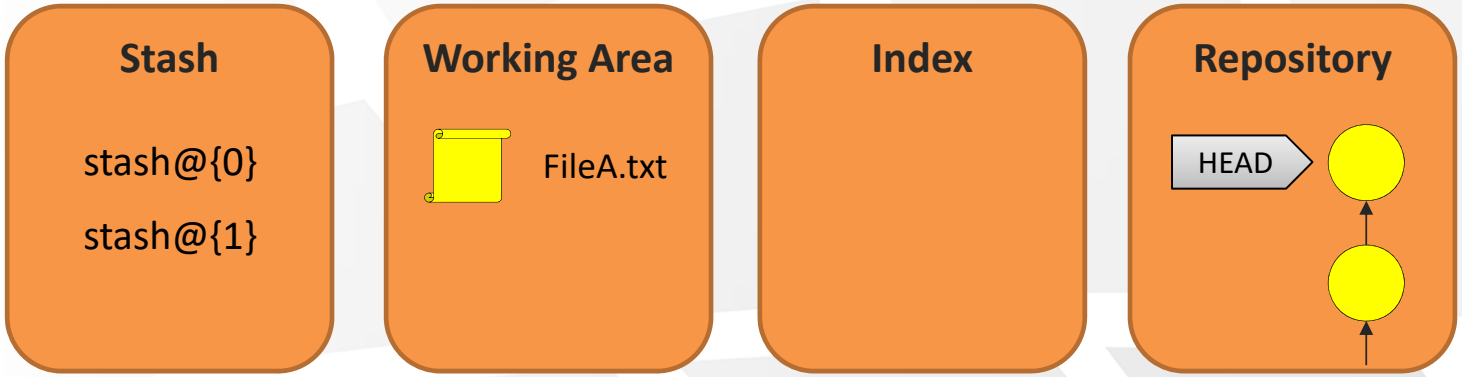

```
$ git stash list

  stash@{0}: WIP on master: c85d13a commit
  stash@{1}: WIP on master: do3dsj5 commit
```

Working Locally (git stash apply)



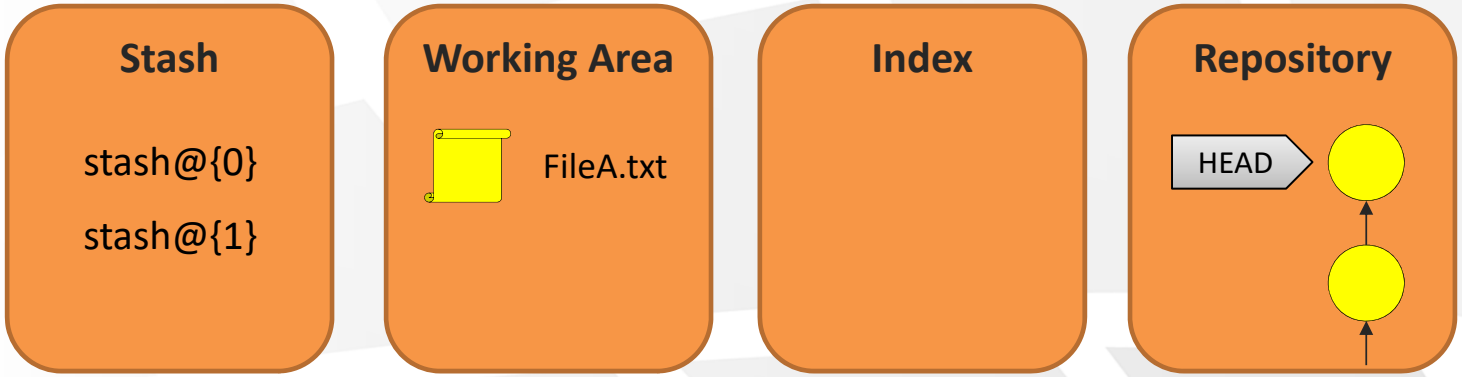

```
$ git stash apply stash@{0}

  On branch master
  Changes to be committed:
  ...
```

Working Locally (git stash drop)




```
$ git stash drop stash@{1}

  Dropped stash@{1} (cdd534f924969bf6d09582bdd48eeafbb689a134)
```

Working Locally (`git stash clear`)

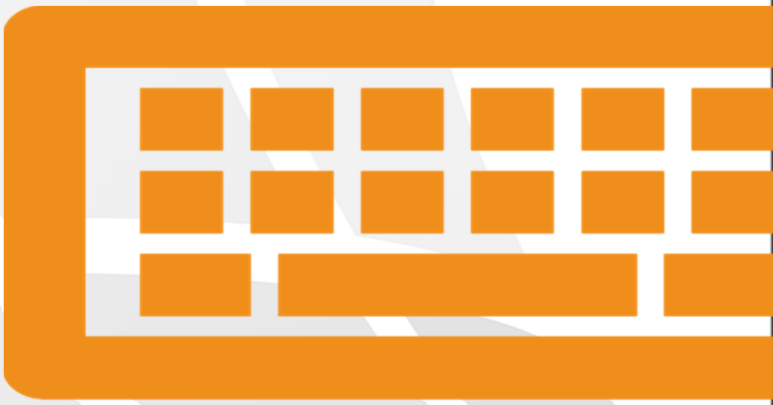



```
$ git stash clear
```

Questions

Lab 6: Stashing Changes

Lab



https://gitlab.com/git-getstarted/lab6

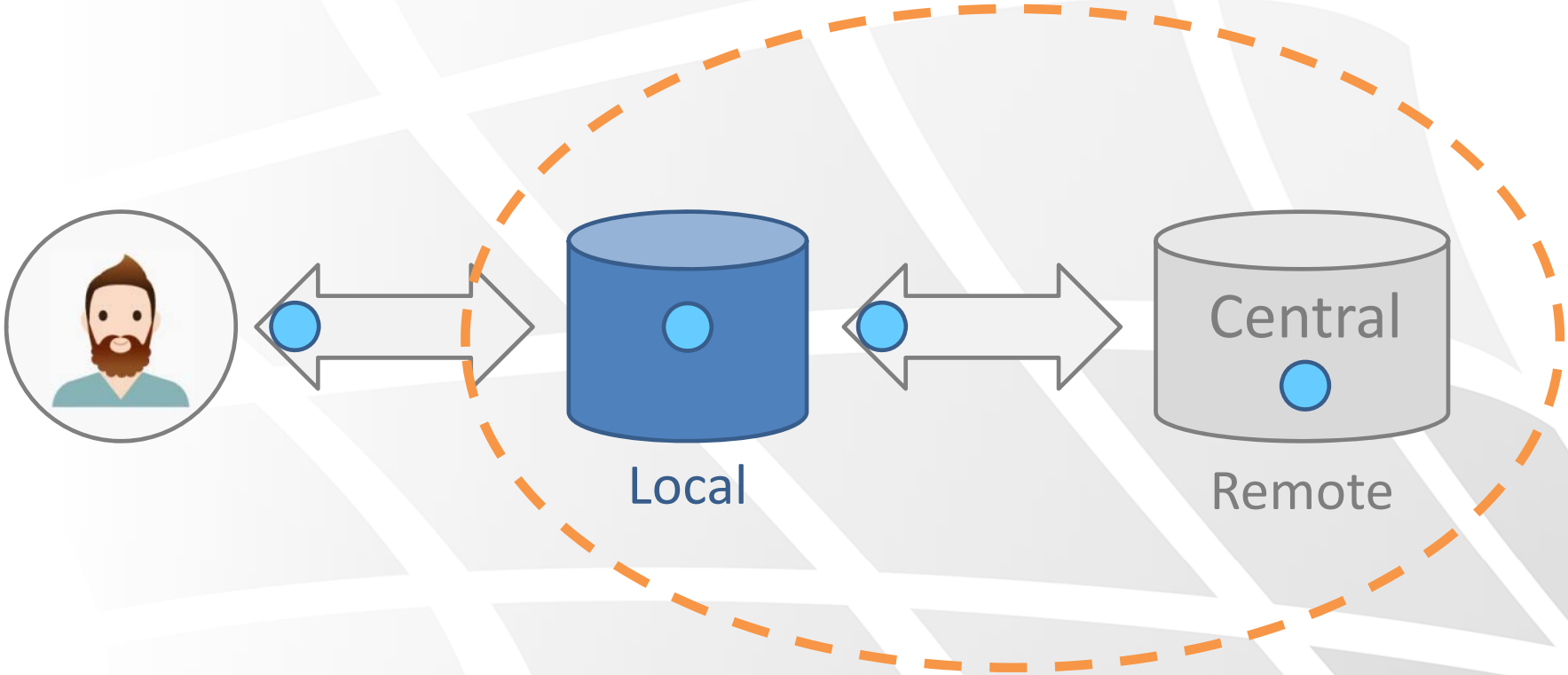
Module 08: Working with Remotes

Get started with Git

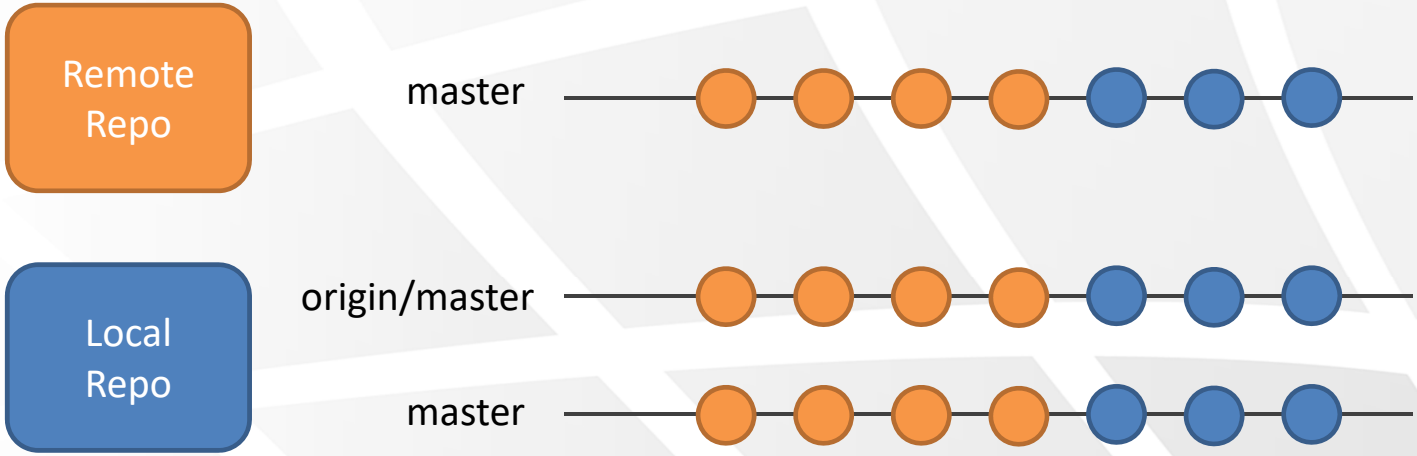
Agenda

- Working with Remotes
- Commands:
  - $ git push
  - $ git fetch
  - $ git pull
- Pull Conflicts
- Lab 7: Working with Remotes

Summary

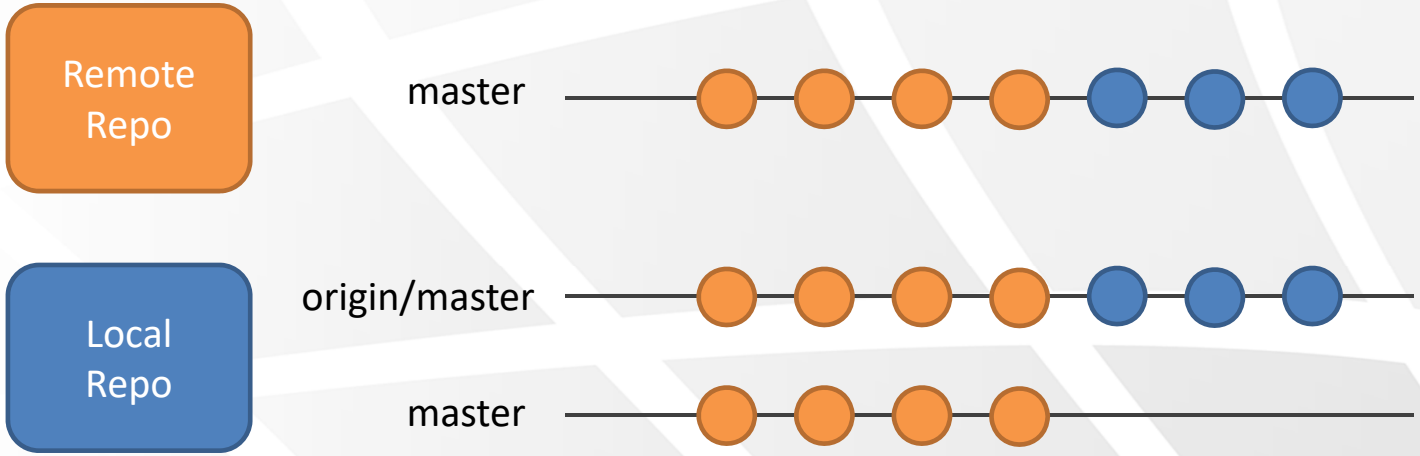


Working Locally (`git push`)




```
$ git push origin master
   Counting objects: 3, done.
   Writing objects: 100% (3/3), 243 bytes | 0 bytes/s, done.
   52938bc..a330c52  master -> master
```

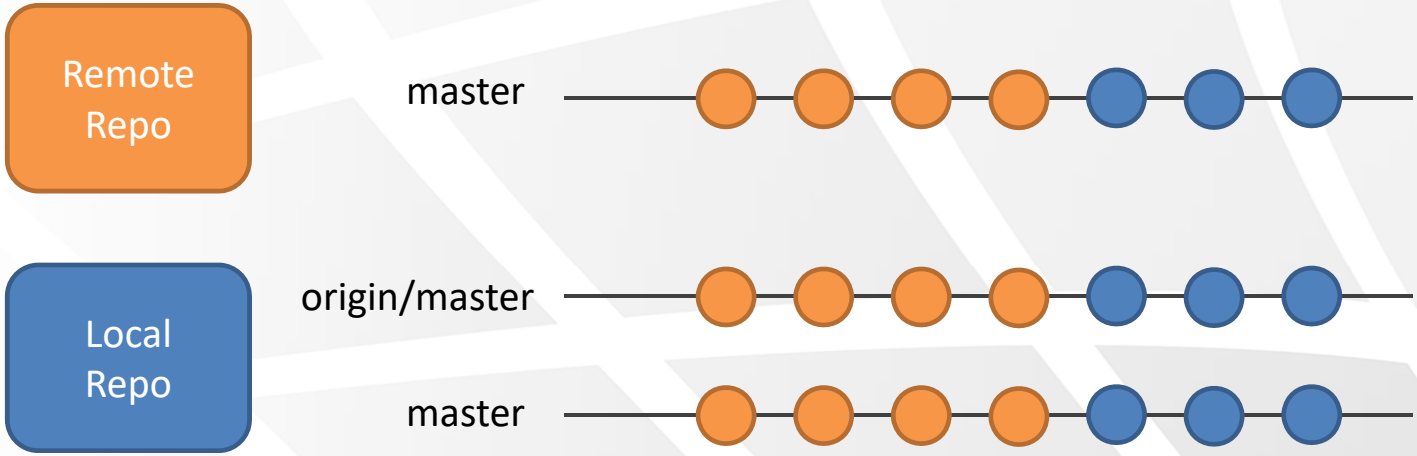
Working Locally (git fetch)




```
$ git fetch origin master
   From https://leonj.visualstudio.com/DefaultCollection/Repo/_git/Test
    * branch            master     -> FETCH_HEAD
      14a2e05..a2e111a  master     -> origin/master
```

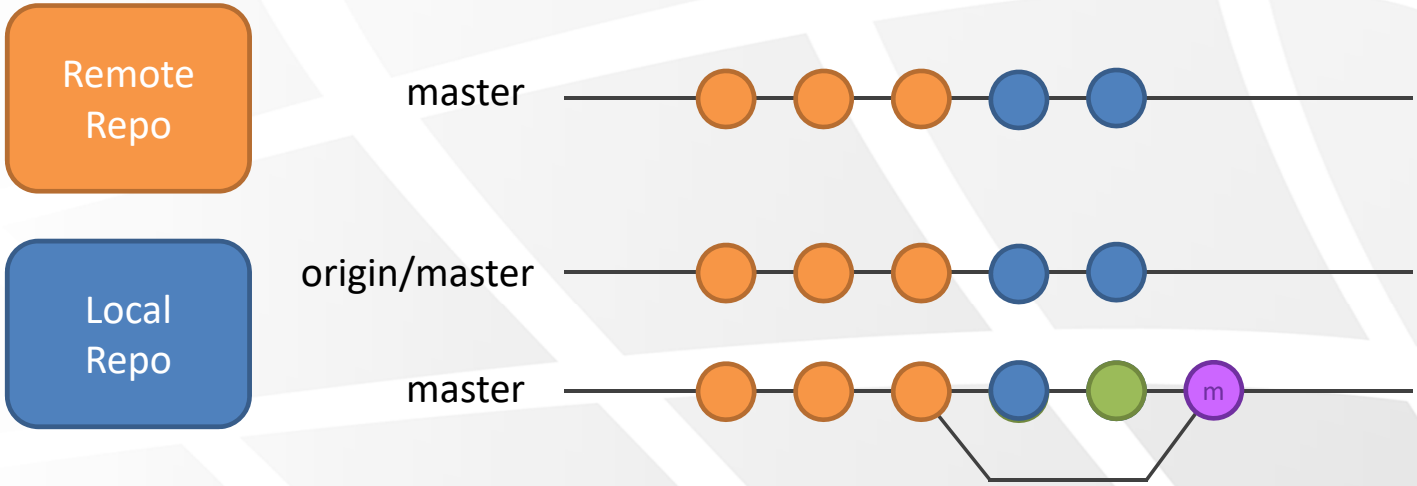
Working Locally `(git pull)`

(fetch + merge)




```
$ git pull origin master
    * branch            master     -> FETCH_HEAD
   Updating 60e69bb..a2e111a
   Fast-forward
```

Working Locally (git pull + conflicts)

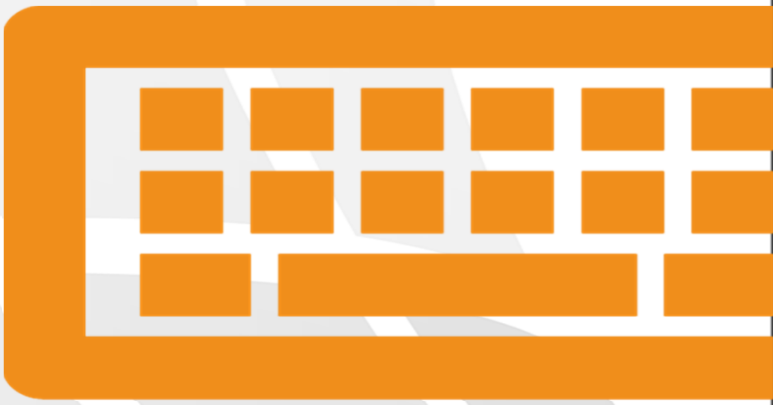



```
$ git pull origin master
   a2e111a..0532902  master     -> origin/master
  Auto-merging File.txt
  Merge made by the 'recursive' strategy.
```

Questions

Lab 7: Working with Remotes

Lab



https://gitlab.com/git-getstarted/lab7

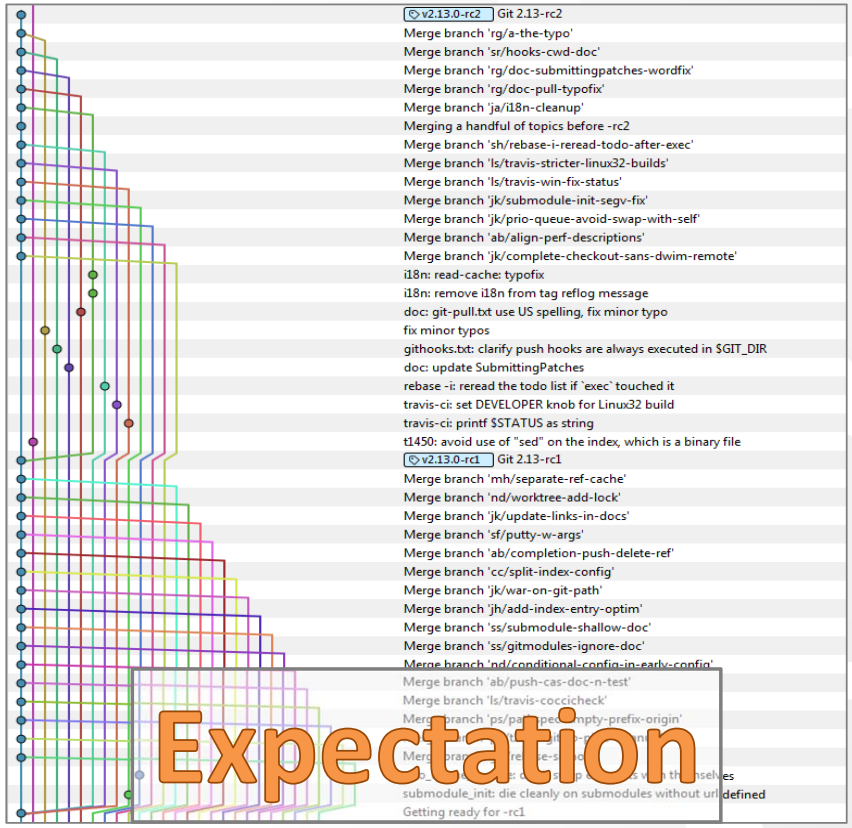
Module 09: Git Workflows

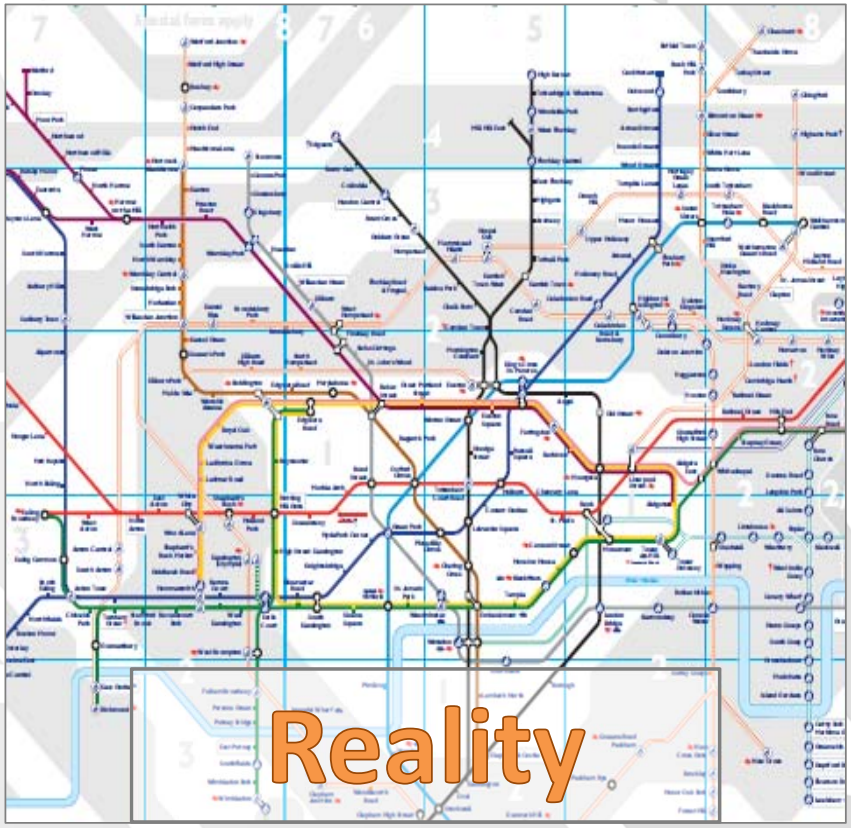
Get started with Git

Agenda

- Introduction
- What is a Git Workflow?
- How Build a Git Workflow:
  - Distribution Models
  - Branches Models
  - Constraints
- Most Known Workflows
- Lab 8: Using a Git Workflow

Introduction – Expectation VS Reality





Git Workflows

- A workflow is a set of conventions that defines how the team should use git, generally composed by:
  - Distribution Model
  - Branches Model
  - Constraints
- The goal of a workflow is to facilitate teamwork and keep the repository clean and organized

Git Workflows

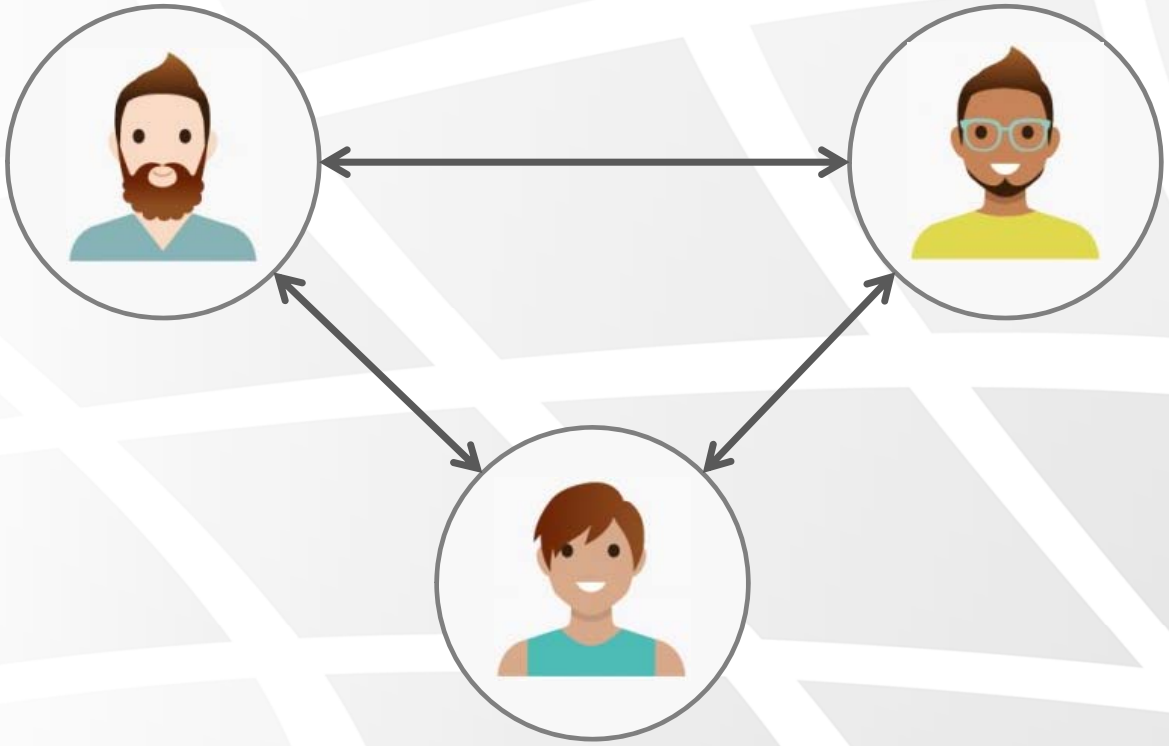
- Lets analyze the pieces that can be used to design our own workflow…

- However decide how to put the pieces together is up to you…

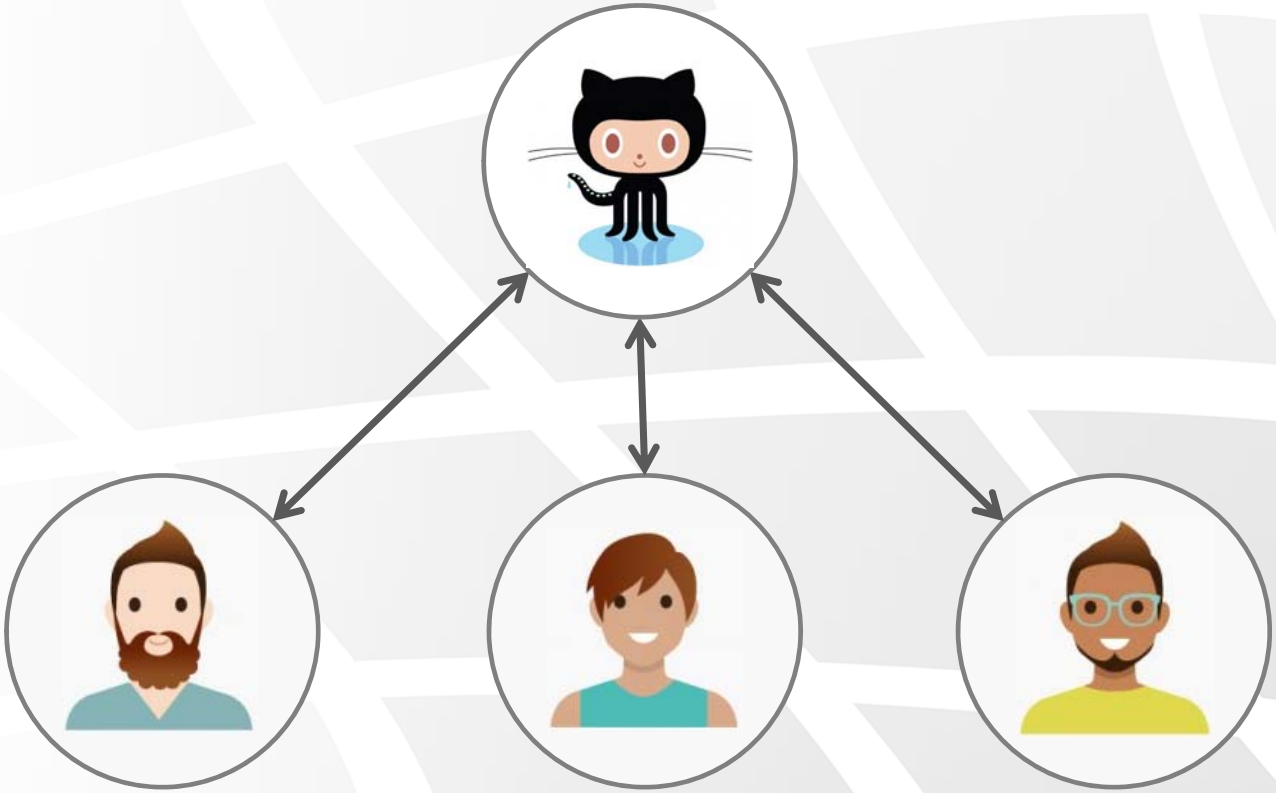
Distribution Models

- How many repositories do you have?
- What will each repository be used for?
- Where will each repository be hosted?
- Who can read/write in each repository?
- Who will manage each repository?

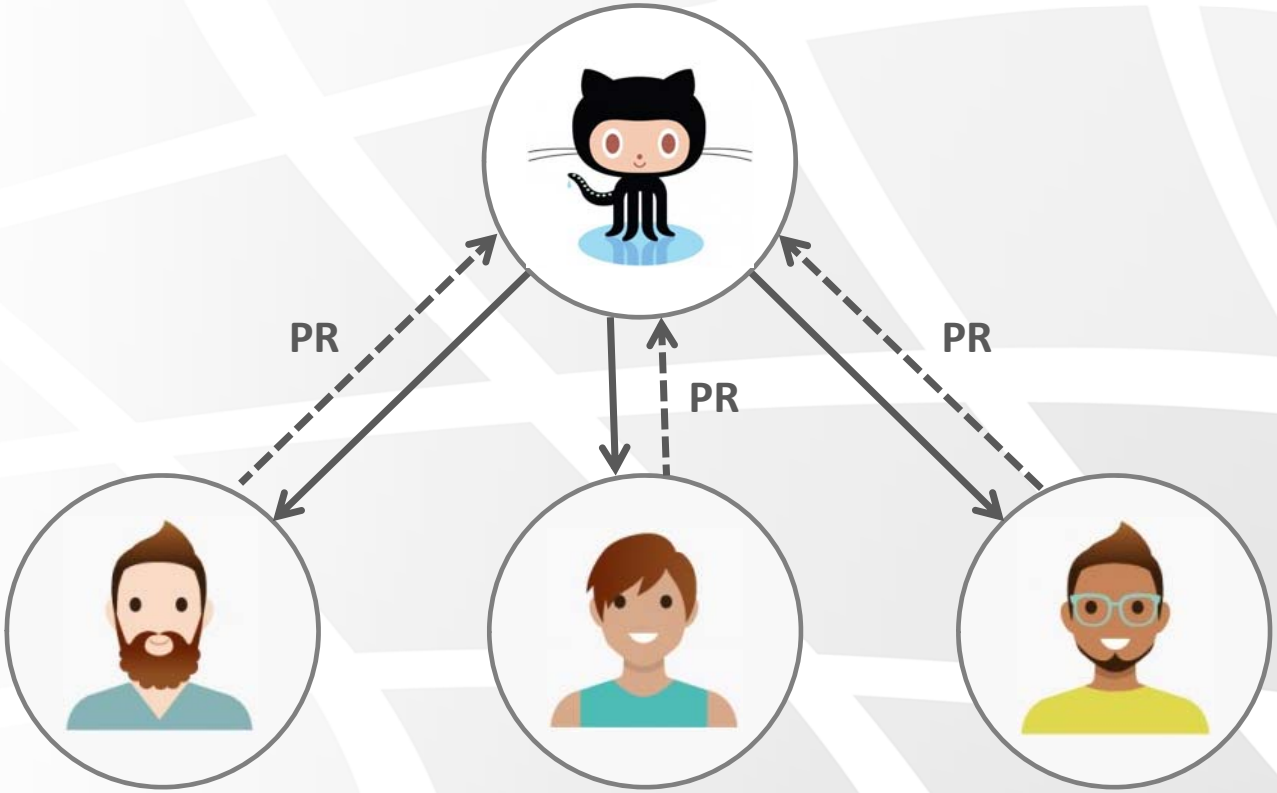
Peer to Peer Model



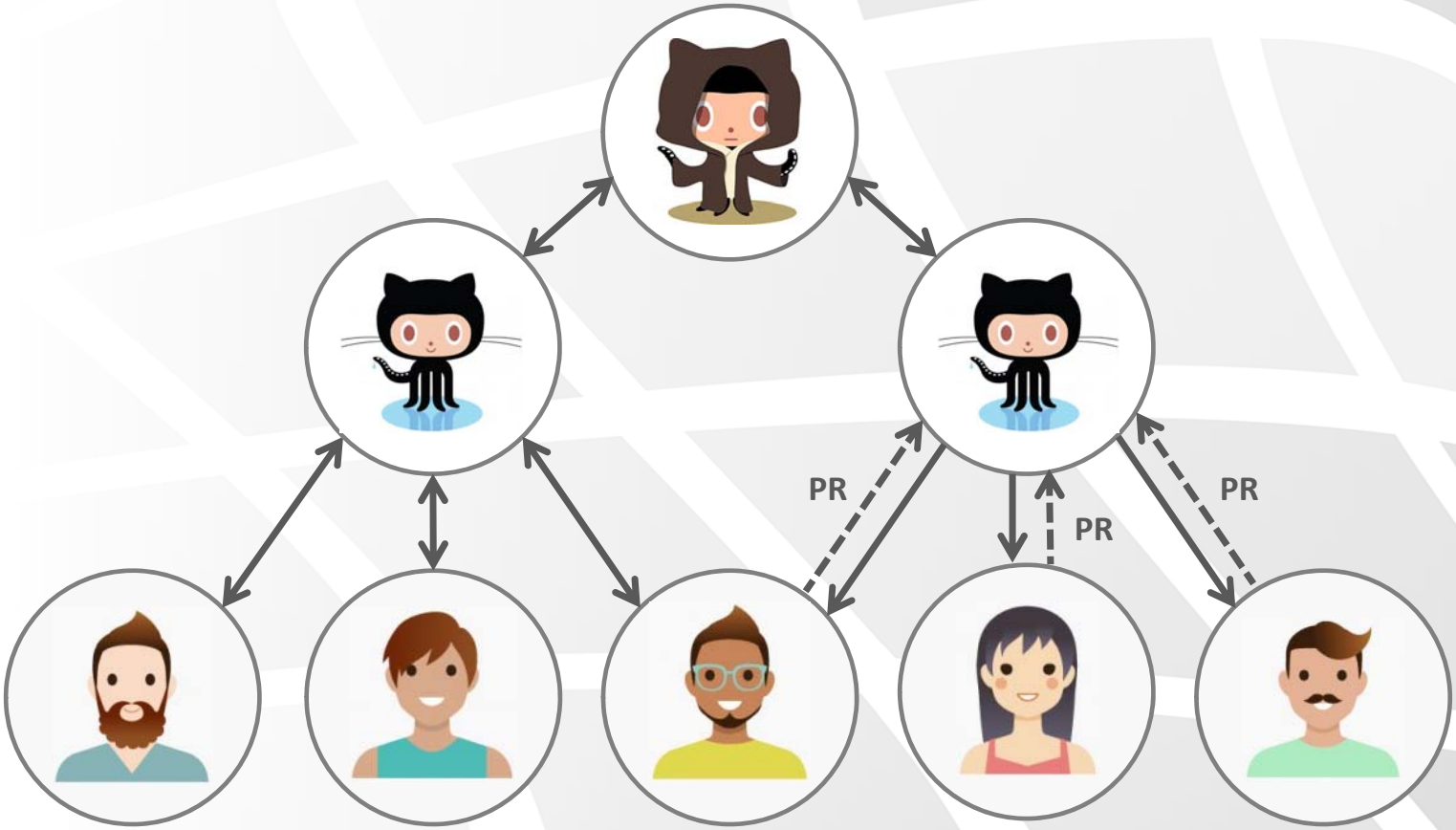
Centralized Model



Pull Request Model



Dictator and Lieutenants Model



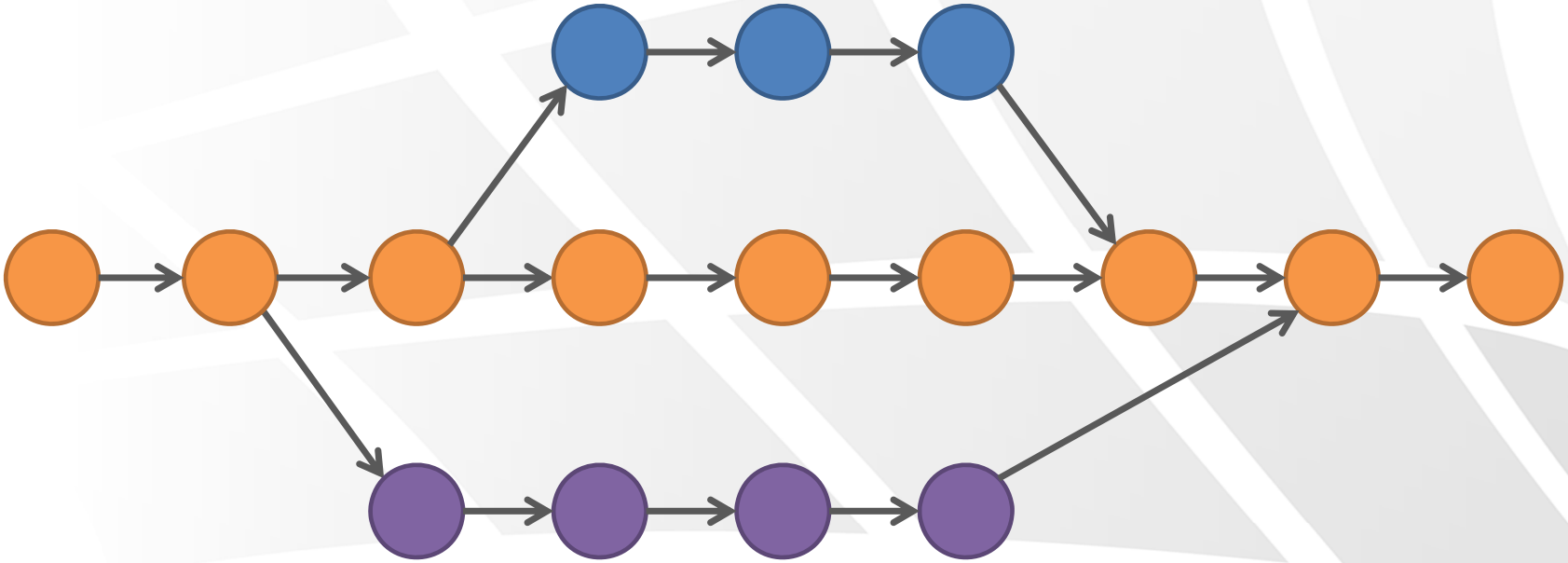
Branching Models

- Which branches do you have?
- How do you use them?
- Who can access each branch?
- Which branches should be merged and when?

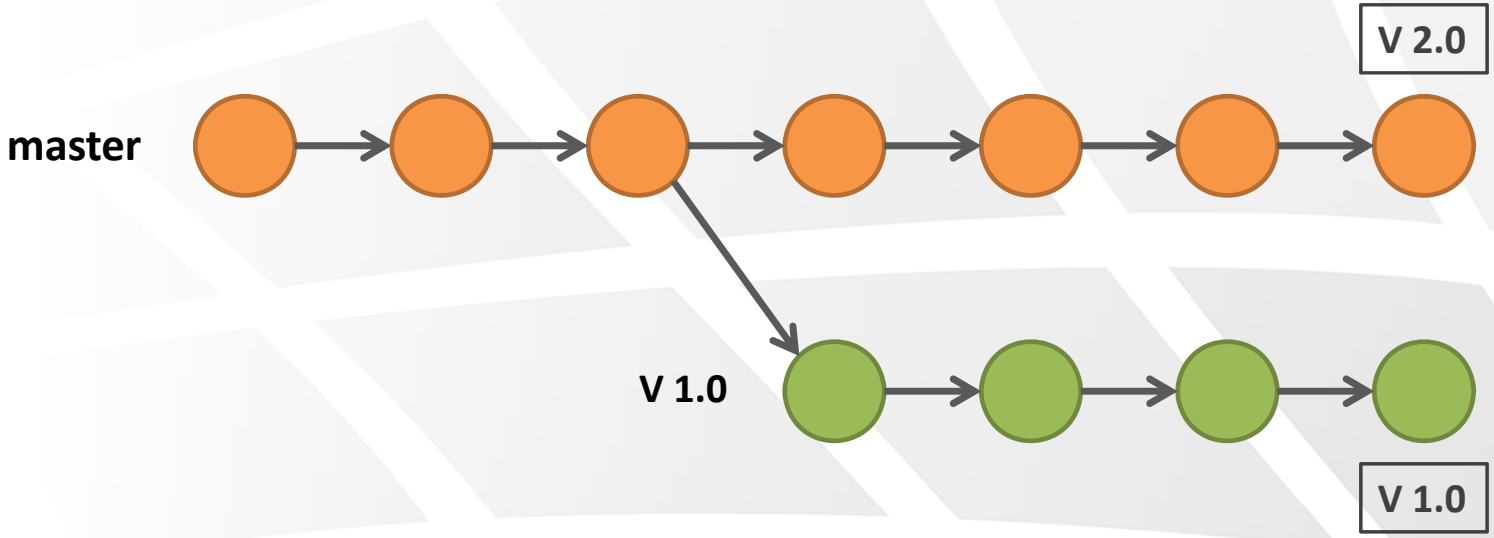
Stable Branch

Unstable Branch

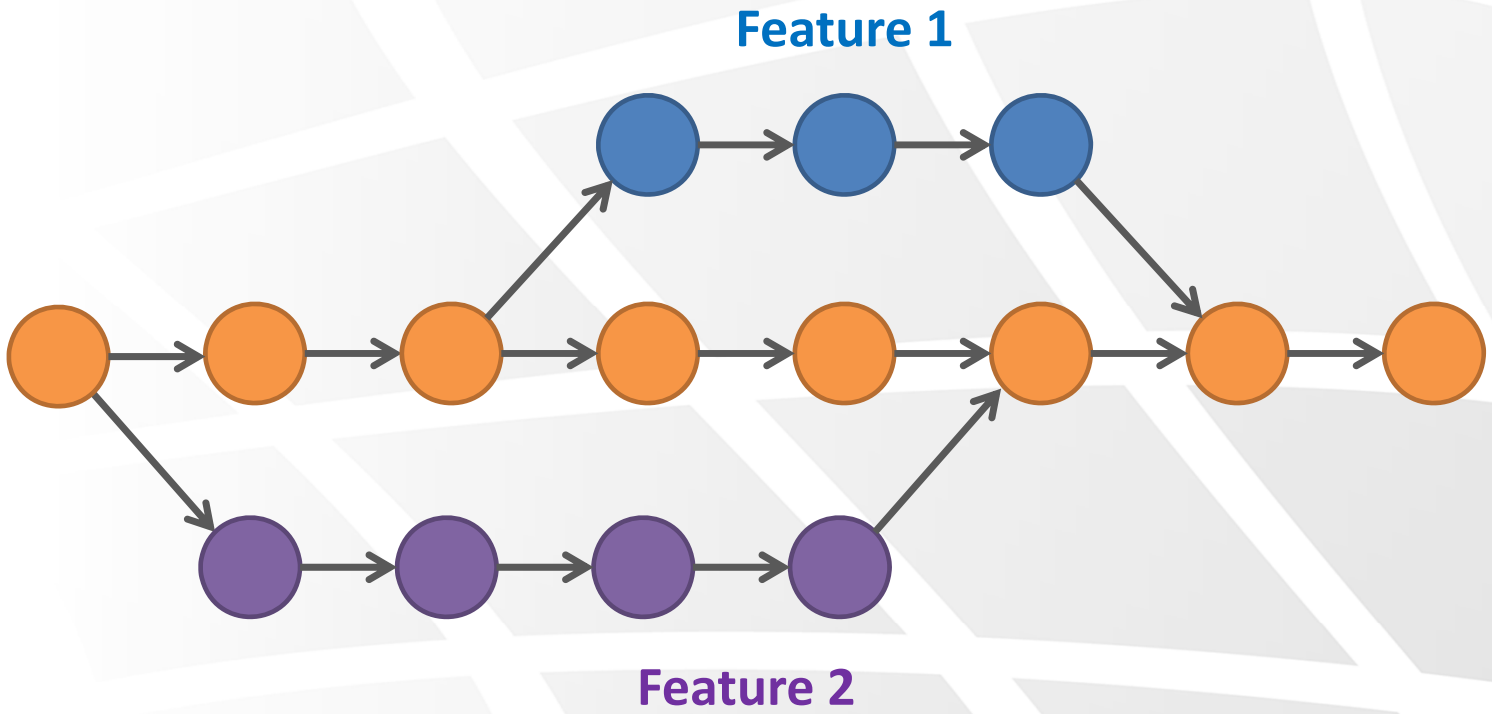
Integration Branch



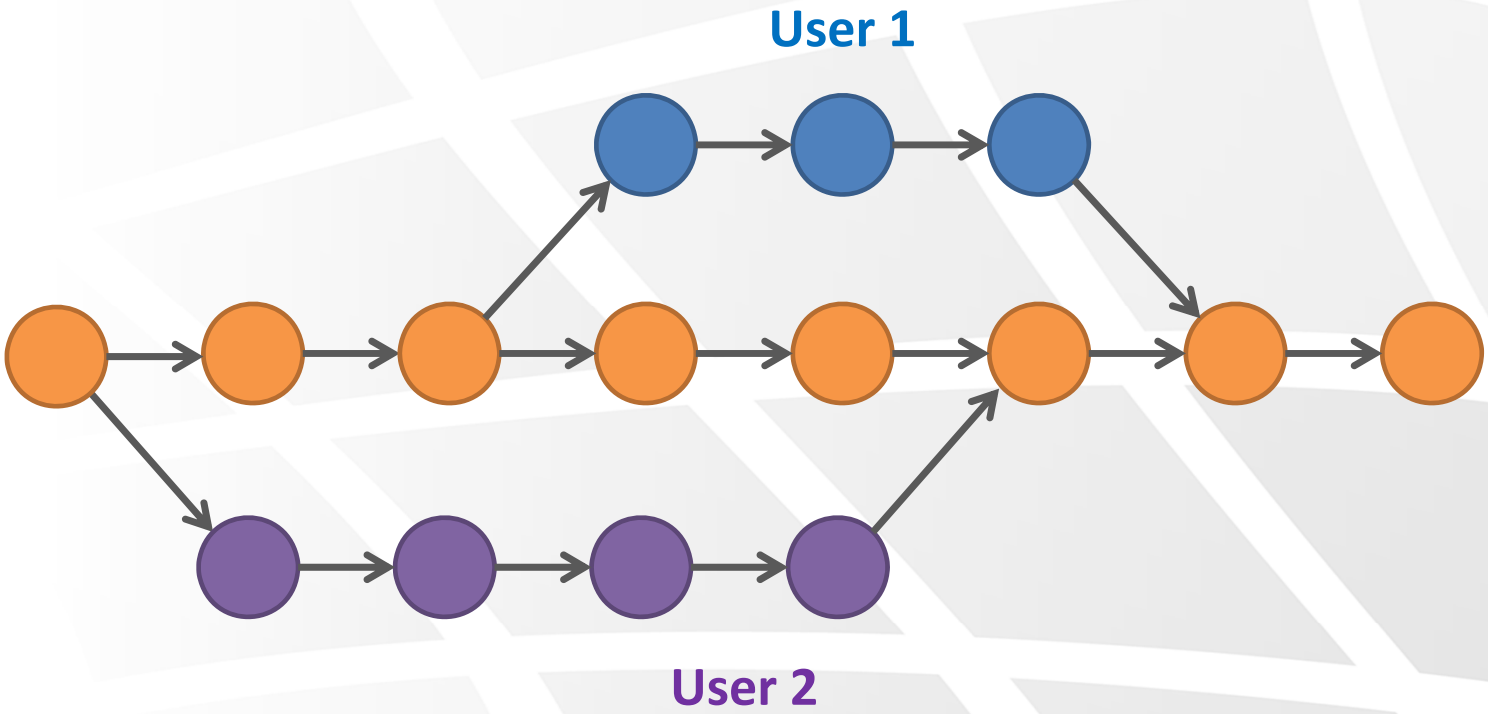
Release Branch



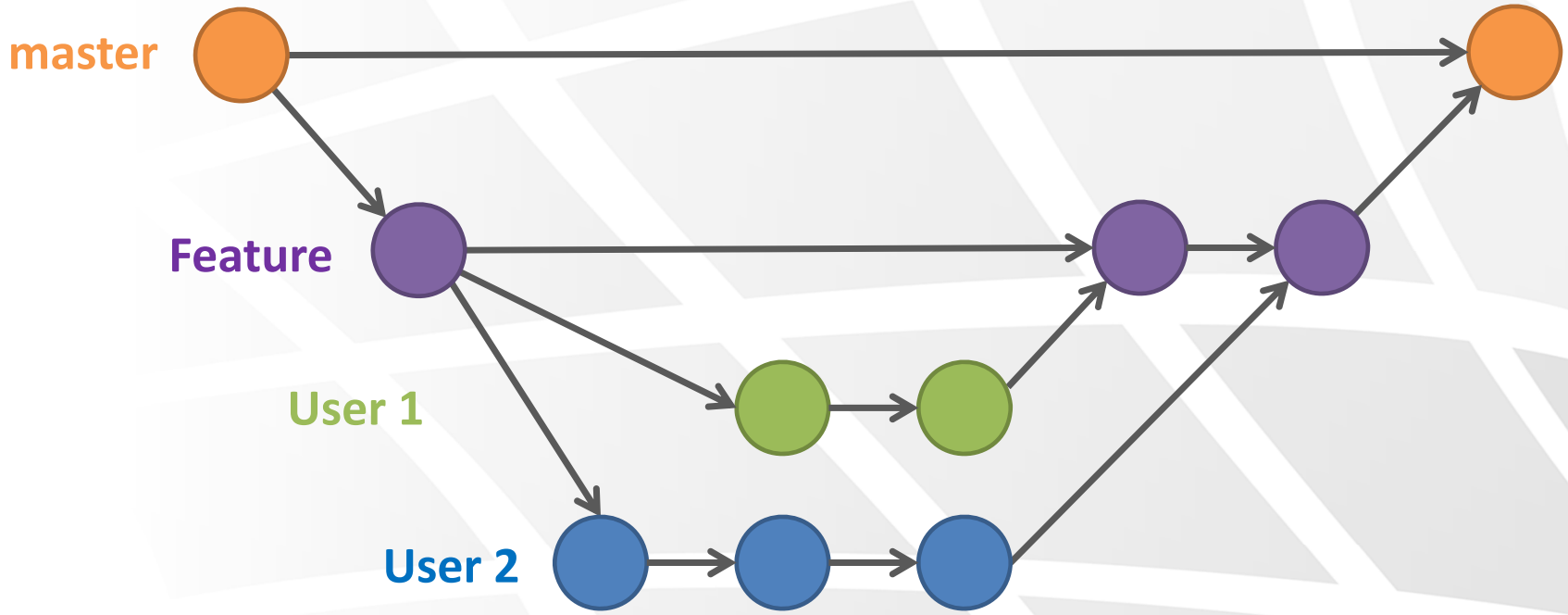
Feature Branch



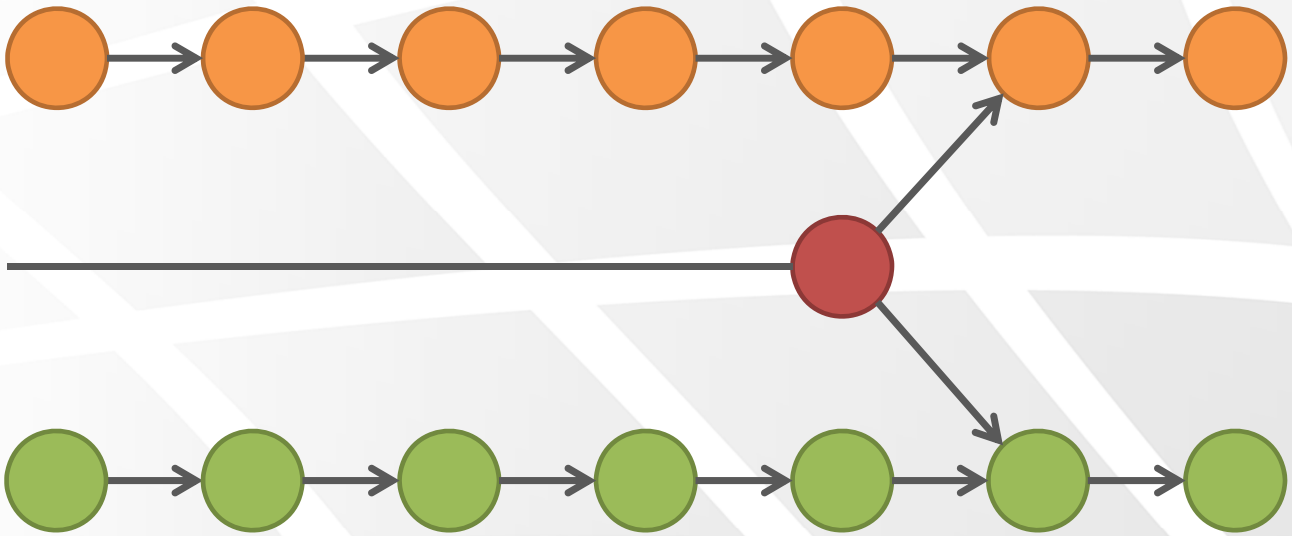
User Branch



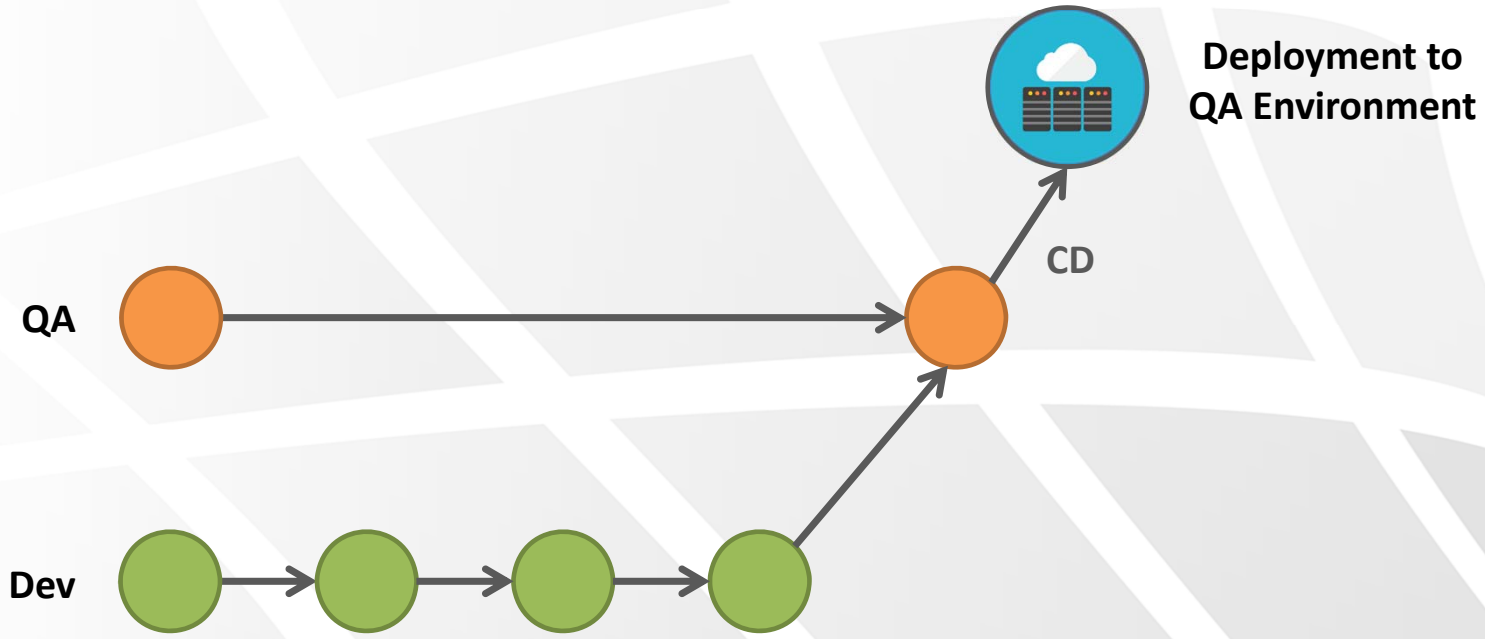
Feature + User Branch



Hotfix Branch



Environment Branch



Constraints

- Do you merge or do you rebase?
- Can you push unstable code? Where yes and where not?
- You will use .gitignore file? For which files?
- Which merge strategy you will use?
- Where and when should tags be used?

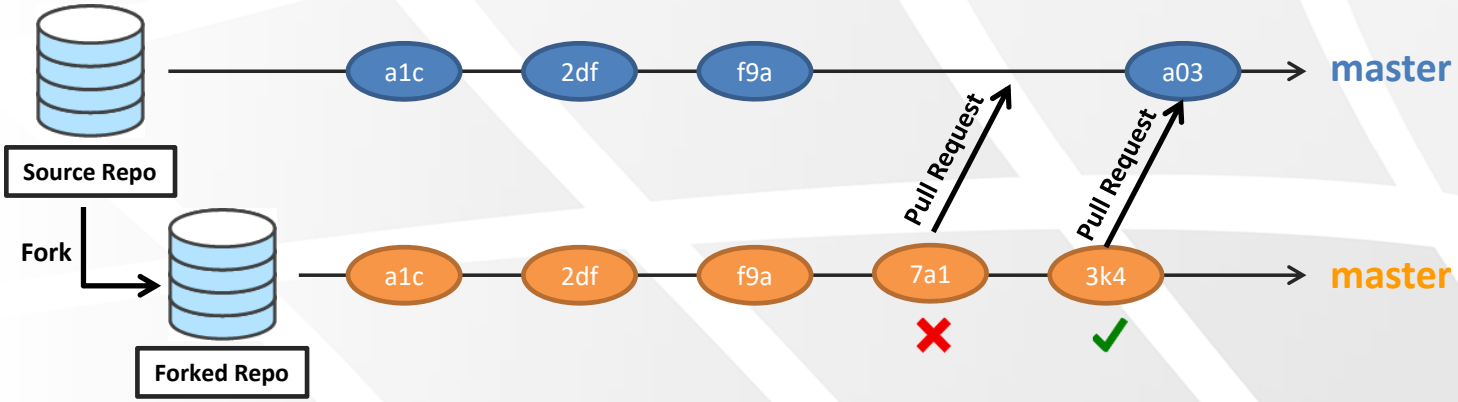
Constraints

- Merge VS Rebase
- Only certain people can do certain things
- Always merge using --no-ff
- Don’t push to unstable branch
- Squash features before merge
- Tag bug fix commits

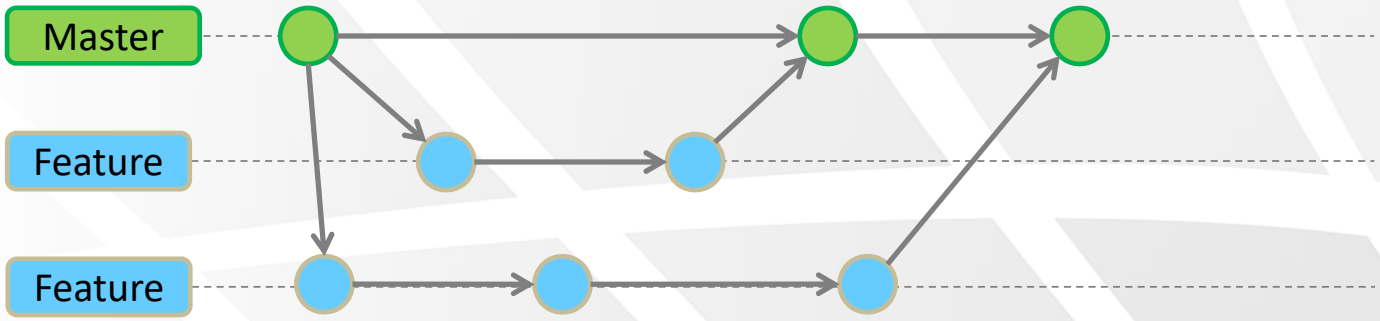
Most Known Workflows

- Forking Workflow
- Feature Branch Workflow
- Environment Workflow
- GitFlow Workflow

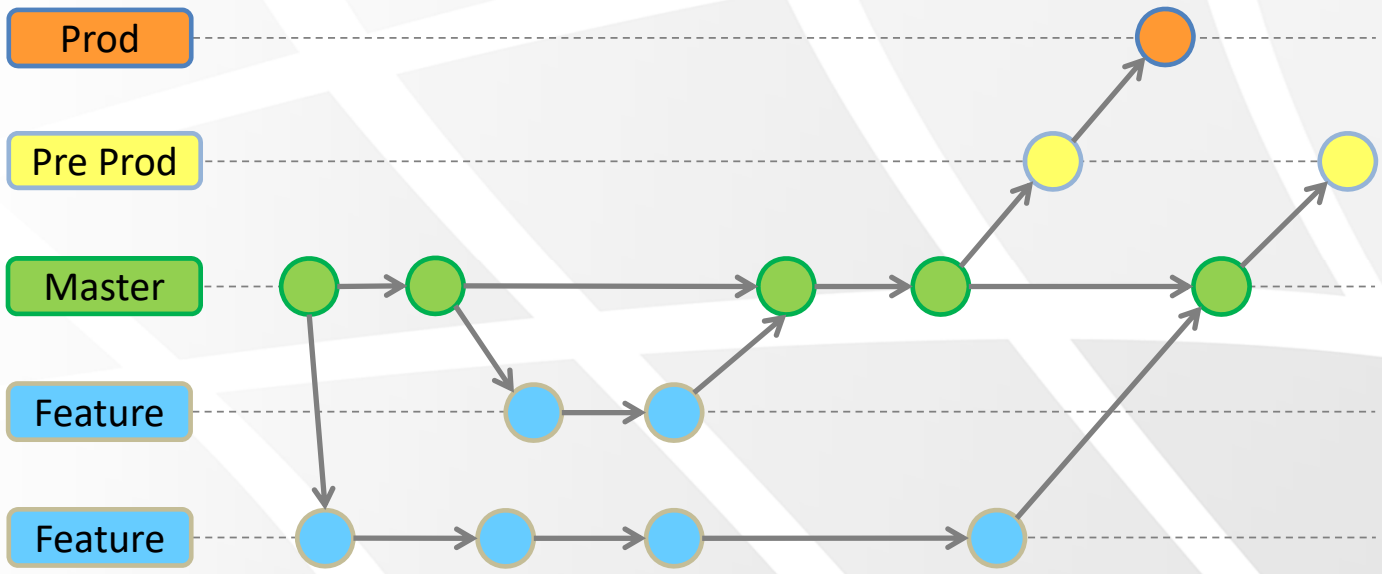
Forking Workflow



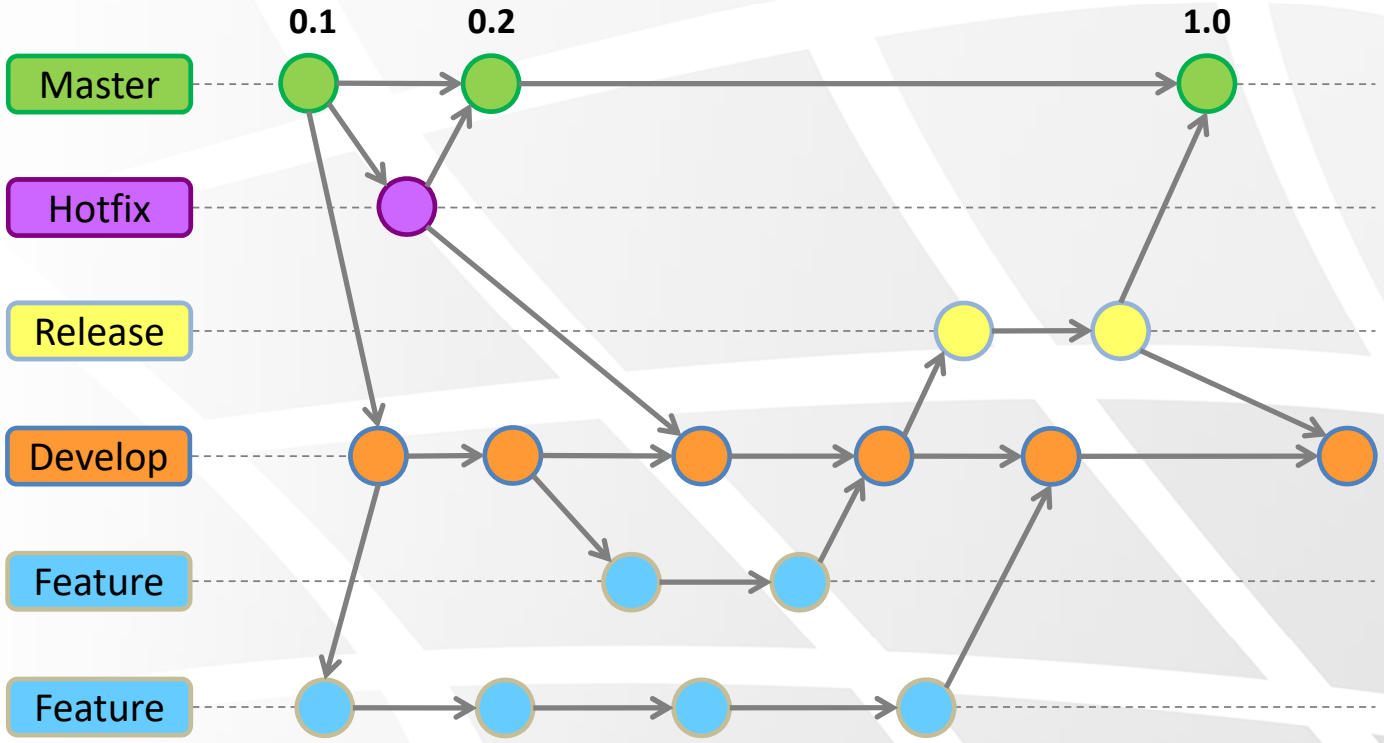
Feature Branch Workflow



Environment Workflow



GitFlow Workflow



Creating a Custom Workflow

- There is no single "correct" workflow
- The most complete workflow is not necessarily the best for you
- The simpler and easy workflow the better
- Don’t look for a workflow that suits your need, create it
- Take into consideration that your needs may change over the time
- Usually the workflow also changes in order to fit your needs

Summary

A Git workflow is the methodology that define the distribution model, the branching model and the constraints for a Git project.

Don’t design a complex workflow

Instead, grow it…

Questions

Module 12: What Next?

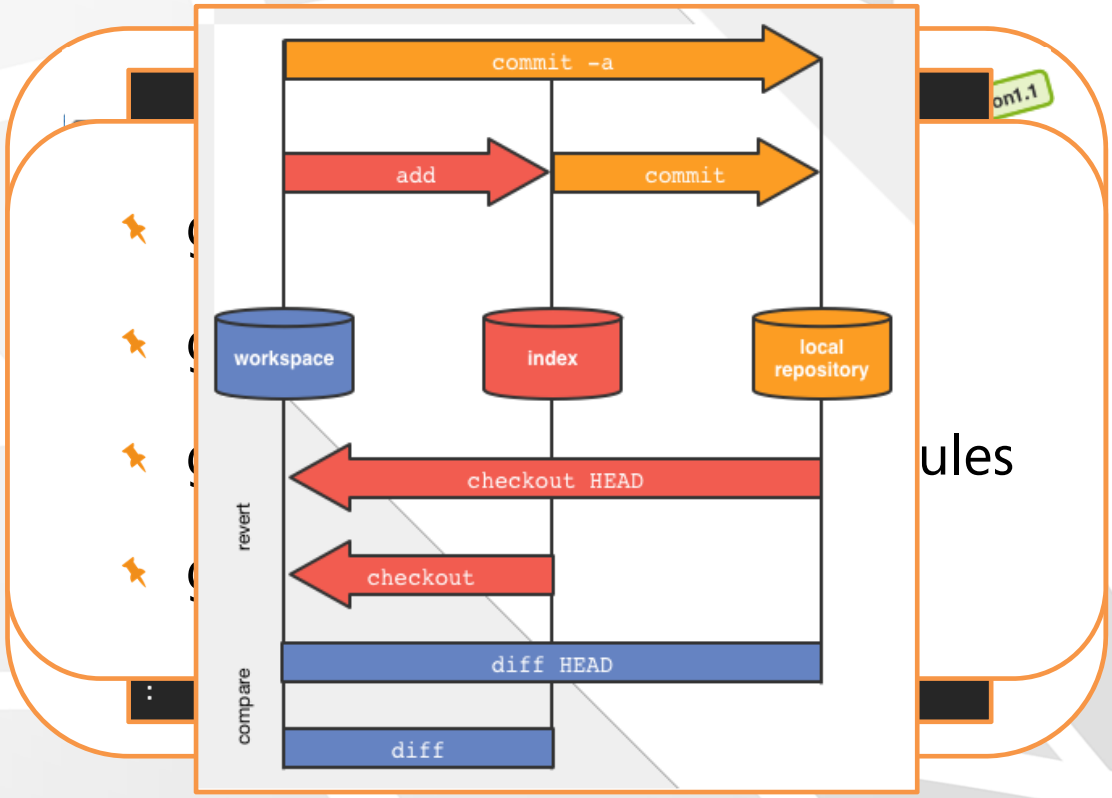
Get started with Git

Agenda

- Course Summary
- What Next?

Course Summary

- Git Structure
- Working Locally
- Working with Remotes
- Git Workflows



What Next?

Get a Cheat Sheet

https://www.git-tower.com/blog/git-cheat-sheet/?utm_source=hashnode.com

https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

https://www.atlassian.com/dam/jcr:8132028b-024f-4b6b-953e-e68fcce0c5fa/atlassian-git-cheatsheet.pdf

https://gitlab.com/gitlab-com/marketing/raw/master/design/print/git-cheatsheet/print-pdf/git-cheatsheet.pdf

four nova

GIT CHEAT SHEET

presented by TOWER › Version control with Git - made easy

CREATE

Clone an existing repository
`$ git clone ssh://user@domain.com/repo.git`

Create a new local repository
`$ git init`

LOCAL CHANGES

Changed files in your working directory
`$ git status`

Changes to tracked files
`$ git diff`

Add all current changes to the next commit
`$ git add .`

Add some changes in <file> to the next commit
`$ git add -p <file>`

Commit all local changes in tracked files
`$ git commit -a`

Commit previously staged changes
`$ git commit`

Change the last commit
Don't amend published commits!
`$ git commit --amend`

COMMIT HISTORY

Show all commits, starting with newest
`$ git log`

Show changes over time for a specific file
`$ git log -p <file>`

Who changed what and when in <file>
`$ git blame <file>`

BRANCHES & TAGS

List all existing branches
`$ git branch -av`

Switch HEAD branch
`$ git checkout <branch>`

Create a new branch based on your current HEAD
`$ git branch <new-branch>`

Create a new tracking branch based on a remote branch
`$ git checkout --track <remote/branch>`

Delete a local branch
`$ git branch -d <branch>`

Mark the current commit with a tag
`$ git tag <tag-name>`

UPDATE & PUBLISH

List all currently configured remotes
`$ git remote -v`

Show information about a remote
`$ git remote show <remote>`

Add new remote repository, named <remote>
`$ git remote add <shortname> <url>`

Download all changes from <remote>, but don't integrate into HEAD
`$ git fetch <remote>`

Download changes and directly merge/integrate into HEAD
`$ git pull <remote> <branch>`

Publish local changes on a remote
`$ git push <remote> <branch>`

Delete a branch on the remote
`$ git branch -dr <remote/branch>`

Publish your tags
`$ git push --tags`

MERGE & REBASE

Merge <branch> into your current HEAD
`$ git merge <branch>`

Rebase your current HEAD onto <branch>
Don't rebase published commits!
`$ git rebase <branch>`

Abort a rebase
`$ git rebase --abort`

Continue a rebase after resolving conflicts
`$ git rebase --continue`

Use your configured merge tool to solve conflicts
`$ git mergetool`

Use your editor to manually solve conflicts and (after resolving) mark file as resolved
`$ git add <resolved-file>`
`$ git rm <resolved-file>`

UNDO

Discard all local changes in your working directory
`$ git reset --hard HEAD`

Discard local changes in a specific file
`$ git checkout HEAD <file>`

Revert a commit (by producing a new commit with contrary changes)
`$ git revert <commit>`

Reset your HEAD pointer to a previous commit
...and discard all changes since then
`$ git reset --hard <commit>`

...and preserve all changes as unstaged changes
`$ git reset <commit>`

...and preserve uncommitted local changes
`$ git reset --keep <commit>`

30-day free trial available at www.git-tower.com

TOWER
Version control with Git - made easy

What Next?

- Explore client tools

- Open a free repository and start coding

What Next?

- Continue learning

  https://try.github.io

  http://learngitbranching.js.org/

  https://www.git-tower.com/learn/

  https://www.atlassian.com/git

  https://www.visualstudio.com/learn-git/

  https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

- Find open source projects and contribute with them

  https://github.com/explore

  https://gitlab.com/explore/projects/starred

  https://bitbucket.org/repo/all

Questions

Thank you for coming!

Get started with Git

Dan Morgenstern

danm@sela.co.il