cisco Live!

# Let's go

The bridge to possible

# Get into Git!

## The Advanced guide to Git commands

Kareem Iskander, Lead Technical Advocate
@Kareem_Isk

DEVNET-2123

# Cisco Webex App

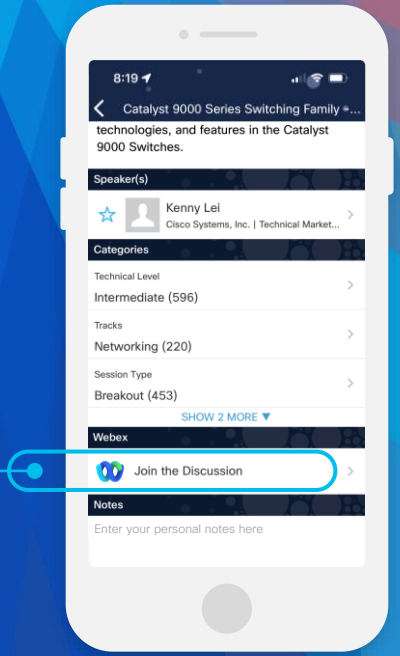
## Questions?

Use Cisco Webex App to chat
with the speaker after the session

## How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

## Webex spaces will be moderated by the speaker until June 9, 2023.




https://ciscolive.ciscoevents.com/ciscolivebot/#DEVNET-2123

# Agenda

- 2min git refresher
- Merging and Pull Request
- Interactive Rebase
- Recovering Deleted Commits
- Submodules
- Search and Find
- Resources

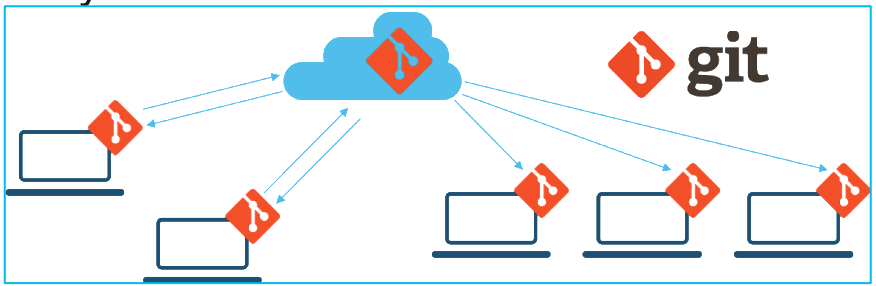
# What is Git?

# Git

- An open-source distributed version control system
- Designed with performance, security, and flexibility in mind
- Stores snapshots of the full file instead of diffs
- Changes are stored in trees
- Trees contain changed files
- Commits contain trees

# Git vs. GitHub

- GitHub is a commercial company, that runs GitHub.com based on Git Version Control System

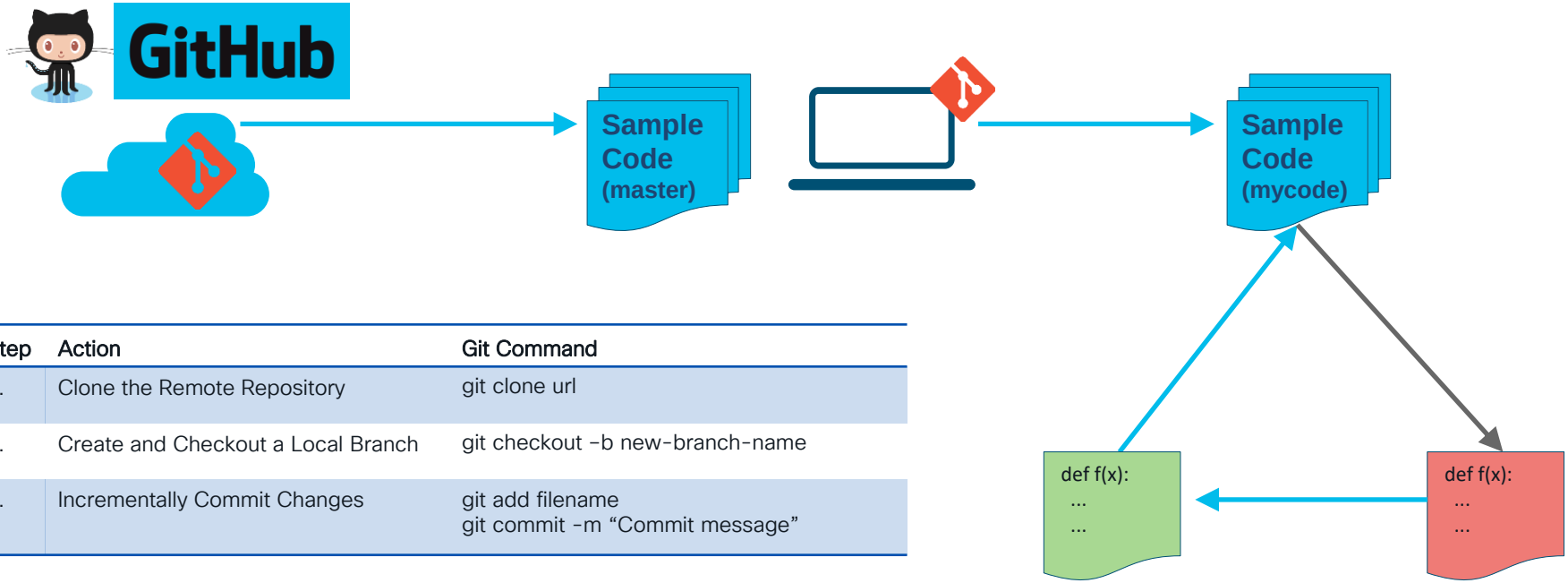



# Git: Technical Overview

# Useful Git Commands

| Action | *What it does* | ***Command*** |
|---|---|---|
| Setup | Tell git who you are<br>*one-time setup* | *> git config --global user.name "your name"*<br>*> git config --global user.email your@email.com* |
| Clone | Clone ("download") a git repository | *> git clone <ur>* |
| Status | Check the Status of your local repository | *> git status* |
| Checkout A Branch | Create and Checkout a local Branch<br>*Creates a "safe place" for your changes* | *> git checkout -b new-branch-name* |
| Add | Add a file to your next commit. | *> git add filename* |
| Commit | Commit your changes. | *> git commit -m "Your commit message."* |
| Checkout A File | Check-out a file from the last commit.<br>*Reverts any changes you have made and restores the last committed version of a file.* | *> git checkout filename* |

# DevNet Sample-Code Workflow




| Step | Action | Git Command |
|---|---|---|
| 1. | Clone the Remote Repository | git clone url |
| 2. | Create and Checkout a Local Branch | git checkout -b new-branch-name |
| 3. | Incrementally Commit Changes | git add filename<br>git commit -m “Commit message” |

# Merge & PR

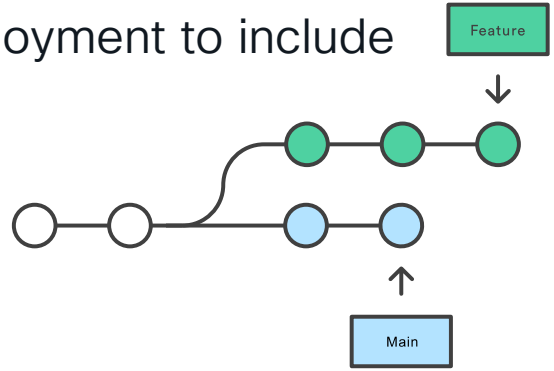
# Merging

- Merging is designed to integrate changes from one branch into another branch

## Use Case

- Multi-site Python deployment automation in the main branch
- The Main branch also contains a newly added Meraki deployment script
- The Feature branch needs to expand on Meraki deployment to include camera option for the new site

A forked commit history

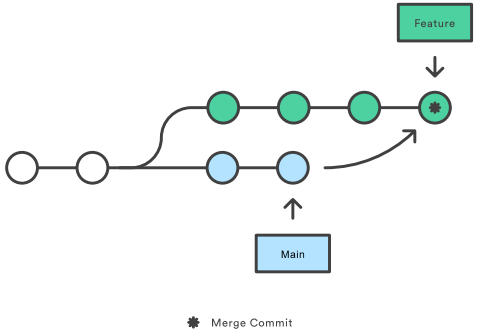


# Merge - Steps

```
git checkout feature
git merge main
```

OR

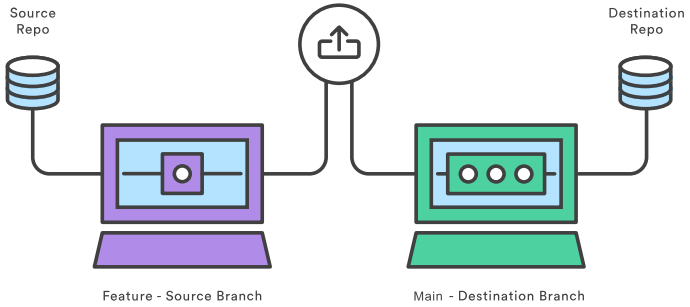
```
git merge feature main
```




- Merging is  a non-destructive operation
- Existing branches are not changed in any way
- **Feature** branch will have irrelevant commits
- Can be difficult for developers to understand features added to branch

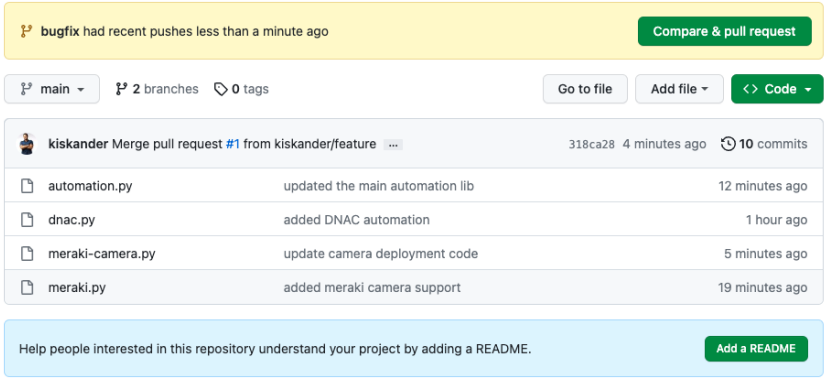
# Pull Request

- Pull requests are a mechanism for a developer to notify team members that they have completed a feature.
- Once their feature branch is ready, the developer files a pull request via their GitHub account
- This lets everybody involved know that they need to review the code and merge it into the main branch.



# Pull Request - Steps

```
git push origin feature
```




## Use Case

- Let's take our Meraki cameras **Feature** branch
- I've made changes, commits and I'd like to notify **Main** branch owners
- **Feature** branch is to become the base code in **Main**

# Let’s try it!

# Interactive Rebase

# Interactive Rebase

## A Tool for Optimizing & Cleaning up Commit Histo

- Change a commit's message
- Delete a commit
- Reorder commits
- Combine multiple commits into one
- Edit/Split an existing commit into multiple new ones

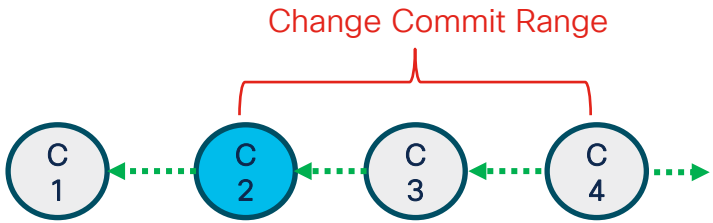
⚠ DO NOT use Interactive Rebase on commits that you've already pushed/shared on remote repo!

# Interactive Rebase

## Use Case

- You are working on the **Feature** branch to expand Meraki automation
- You have been making commits ever function you write
- You are ready to merge into **Main** branch
- You have realized:
  1. “Over Commit-ted” - get it?
  2. Commit messages aren’t cutting it

# Interactive Rebase - Steps




1. How far back do you want to go?

2.
```
git rebase -i HEAD~2
```

3. Determine <u>action</u> to apply to your commits

4. Make changes & save

5.
```
git log oneline
```

# Let’s try it!

# Cherry Picking

# Cherry Picking

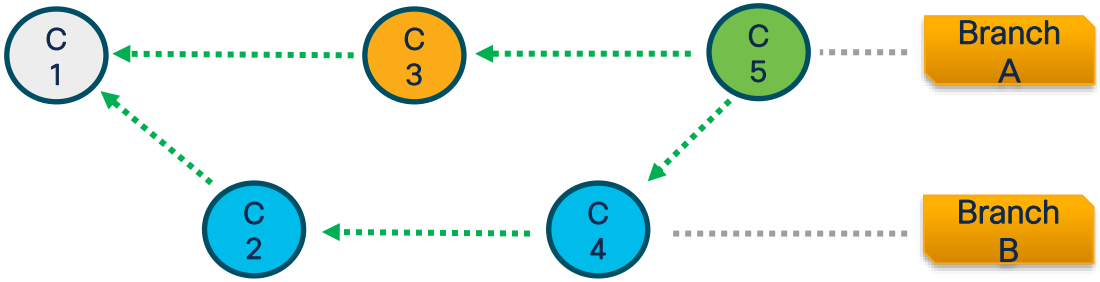
## Use Case

- You are working on the **Feature** branch to expand your Meraki site automation
- You have made a commit on the **Feature** Branch
- You noticed you are working in the **Main** Branch
- The commit does not belong in **Main** .. Yet
- What to do??

# Cherry Picking

## Merging Branches

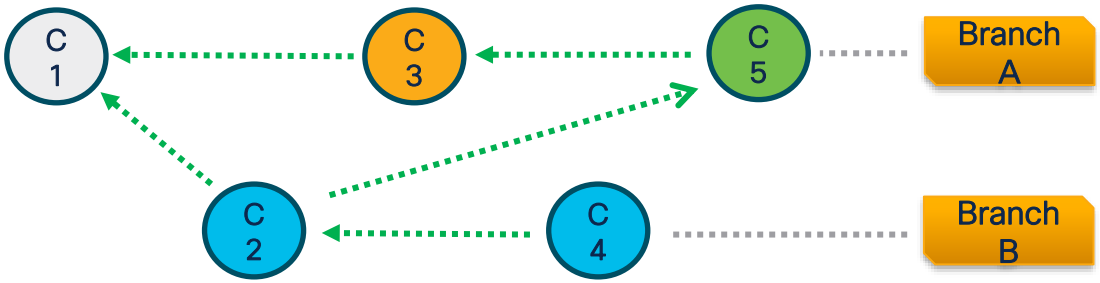
- Merging is designed to integrate changes from one branch into another branch (as we've seen)
- New Commits from **Branch B** is copied over to the HEAD of the **Branch A**



# Cherry Picking

## Integrate Single, Specific commit

- Cherry Picking allows you to select individual commits
- Cherry Picking integrate specific commits to Branch
- Only Commit C2 from Branch B to integrate into Branch A



# Cherry Picking - Steps

1. While on **Main** Grab the commit

```
git log oneline
```

2. ranch

```
git checkout Feature
```

3.

```
git cherry-pick a827df1
```

4. Clean up **Main**

```
git checkout main
git reset --hard HEAD~1
```

# Let’s try it!

# Reference logs

# Reflog

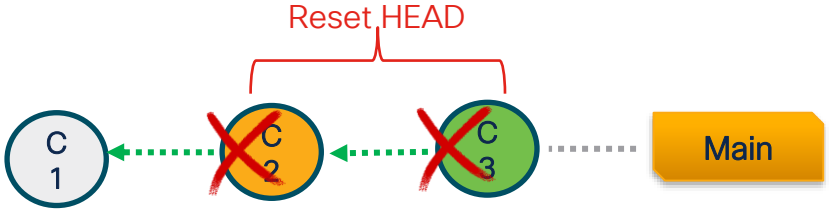
- Reflog is a journal of all the movement
- Reflog is a Protocol of HEAD Pointer Movements
- Merge, Rebase, Cherry Pick, Reset .. All of the movements

```
01-merge-pr
~
~
~
~
kiskande:01-merge-pr/ (main) $ git reflog                                                    [18:33:23]
ee864e7 (HEAD -> main) HEAD@{0}: checkout: moving from bugfix to main
8cf5672 HEAD@{1}: commit: bug fixes
afe5ba3 HEAD@{2}: checkout: moving from feature to bugfix
afe5ba3 HEAD@{3}: commit: added connection string json
0ef3009 HEAD@{4}: checkout: moving from main to feature
ee864e7 (HEAD -> main) HEAD@{5}: checkout: moving from feature to main
0ef3009 HEAD@{6}: commit: update camera deployment code
264dfdc HEAD@{7}: commit: created meraki camera support
ee864e7 (HEAD -> main) HEAD@{8}: checkout: moving from main to feature
ee864e7 (HEAD -> main) HEAD@{9}: merge feature: Fast-forward
e806b9f HEAD@{10}: checkout: moving from feature to main
ee864e7 (HEAD -> main) HEAD@{11}: checkout: moving from main to feature
e806b9f HEAD@{12}: checkout: moving from feature to main
ee864e7 (HEAD -> main) HEAD@{13}: merge main: Merge made by the 'recursive' strategy.
1785fbb HEAD@{14}: checkout: moving from main to feature
e806b9f HEAD@{15}: commit: updated the main automation lib
7816048 HEAD@{16}: checkout: moving from feature to main
1785fbb HEAD@{17}: commit: added meraki camera support
7816048 HEAD@{18}: checkout: moving from main to feature
7816048 HEAD@{19}: commit: added DNAC automation
a827df1 HEAD@{20}: commit: added XE ZTP support
e2e52b1 HEAD@{21}: commit: created and tested automation script
5be16ca HEAD@{22}: commit (initial): init commit of automation.py
(END)
```

# Reflog

## Use Case

- You are working on the **Feature** branch to expand your Meraki site automation
- You have gone “commit-crazy”
- You think you want to get rid of some commits
- You use “git reset”
- You delete an important commit
- Panic Mode 😱



# Reflog - Steps

1. Reflog are in chronological order

```
git reflog --oneline
```

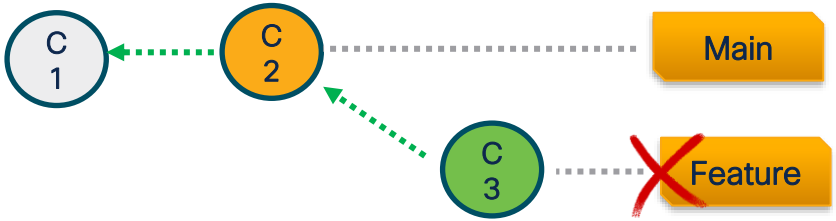
2. Find the catastrophic reset and copy the state before into new Branch

```
git branch recovery 8cf5672
```

# Reflog

## Use Case

- You are working on the **Feature** branch to expand your Meraki site automation
- You think you merged the **Feature** branch, but you didn't
- You delete the branch
- Panic Mode 😱



# Let’s try it!

# Submodules

# Submodules

- Git Repo inside Git Repo
- Submodule is a standard git repo which means you can add, commit, pull ..
- Difference is that it is nested within a parent repo
- Important to know:
  - Submodule content are not stored in the parent repository
  - Parent repo stores:
    - Submodule remote URL
    - Local path
    - Checked out revision

# Submodules

## Use Case

- You decided to use Meraki SDK to automate your Meraki Deployment
- SDK needs to be embedded into your **Branch**
- Option 1 (bad):
  - Download the SDK , Mix it with your code and treat it as one branch
  - Updating the external code is now a manual process
- Option 2 (good):
  - Git Submodules

# Submodules - Steps - Local Project

1. Let's create a subfolder with our library

```
mkdir lib
cd lib
```

2. Grab the library as a Git Submodule

```
git submodule add https://github.com/meraki/dashboard-api-python.git
```

3. View system files

```
cat .gitmodules
cat .git/config
```

⚠ DO NOT forget to commit the submodule in Parent repo

# Submodules - Steps - Cloned Repo

1. Clone Repo with Submodules

```
git clone https://github.com/apache/airflow.git
```

2. Notice the submodules, empty folders

| | |
|---|---|
| actions | Today at 2:46 PM |
| breeze | Today at 2:44 PM |
| action.yml | Today at 2:44 PM |
| build-ci-images | Today at 2:44 PM |
| action.yml | Today at 2:44 PM |

3. Populate the submodules

```
git submodule update --init --recursive
```

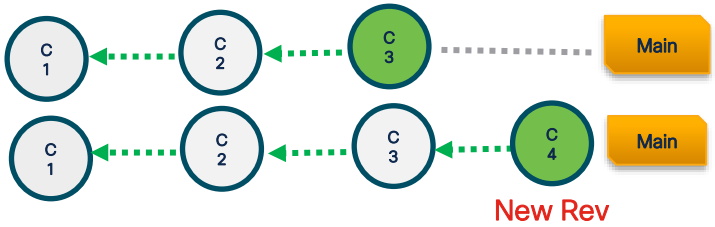
OR

```
git clone --recurse-submodules <url>
```

# Submodules – Revision

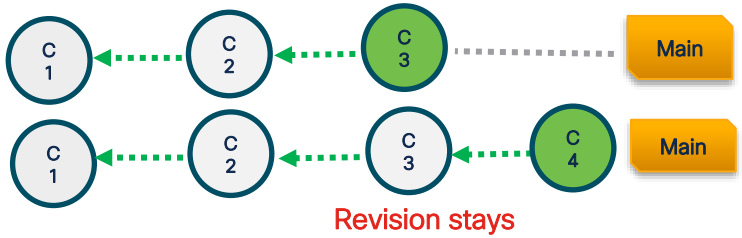
## In a “Standard” Git Repo

- You checkout a Branch
- The last commit is your checkout revision
- The new revisions are always the latest commits




## In a Submodule Git Repo

- Your last commit is your checkout revision
- You must explicitly update and commit a submodule for the point to move



# Let’s try it!

# Search and Find

# Search & Find

| Action | *flag* | ***Command*** |
|---|---|---|
| By date | *--before / -- after* | *git log --after="2023-2-1" --before"2023-4-1"* |
| By message | --grep | *git log --grep="" (Can use Regex)* |
| By author | --author | *git log --author="Kareem iskander"* |
| By file | *-- <filename>* | *git log -- README.md* |
| By branch | <branch-A> | *git log feature..main (in main but not in feature)* |

# Resources

# Cisco U.

Tech learning, shaped to you.

u.cisco.com

# Kareem Iskander

Lead Technical Advocate, Cisco Learning & Certification

**kiskande@cisco.com**

**@Kareem_Isk**

**https://github.com/CiscoLearning**

**https://www.youtube.com/@CiscoUtube**

# Fill out your session surveys!

Attendees who fill out a minimum of four session surveys and the overall event survey will get **Cisco Live-branded socks** (while supplies last)!

Attendees will also earn 100 points in the **Cisco Live Challenge** for every survey completed.


**These points** help you get on the leaderboard and increase your chances of winning daily and grand prizes

# Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

# Thank you

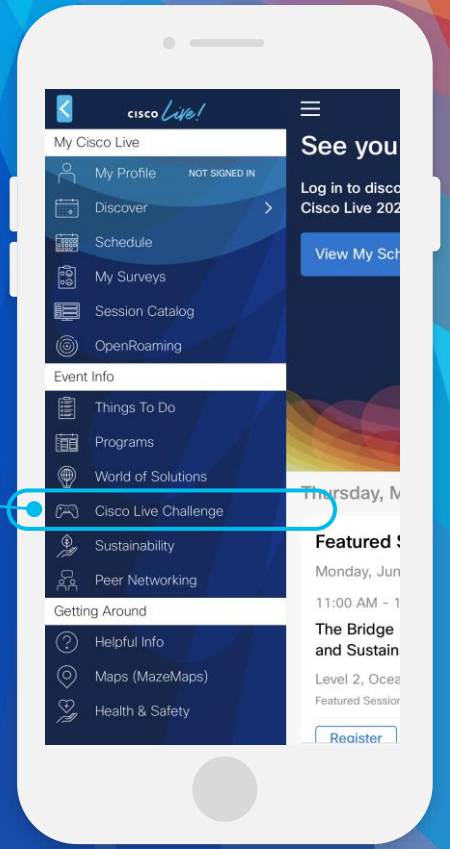
# Cisco Live **Challenge**

## Gamify your Cisco Live experience!

**Get points for attending this session!**

## How:

1. Open the Cisco Events App.
2. Click on 'Cisco Live Challenge' in the side menu.
3. Click on View Your Badges at the top.
4. Click the + at the bottom of the screen and scan the QR code:



cisco Live!

# Let's go