

>>>network.toCode()

Collaborative Workflows with Git and GitHub

Workshop



Introductions

To Network to Code and the Training Library

>>> Who is Network to Code?



Network Automation Solutions Provider

Founded in 2014, we help companies transform the way their networks are deployed, managed, and consumed using network automation and DevOps technologies.



A Diverse Team, with Deep Expertise

Engineers and developers in network automation, software and security, with leadership from vendors, integrators, and top tier consulting firms - all drive value to our clients.



Vendor Neutral Community

Partner with all OEMs, develop solutions with commercial and open source components. Host 19,000+ members and 300+ channels at slack.networktocode.com

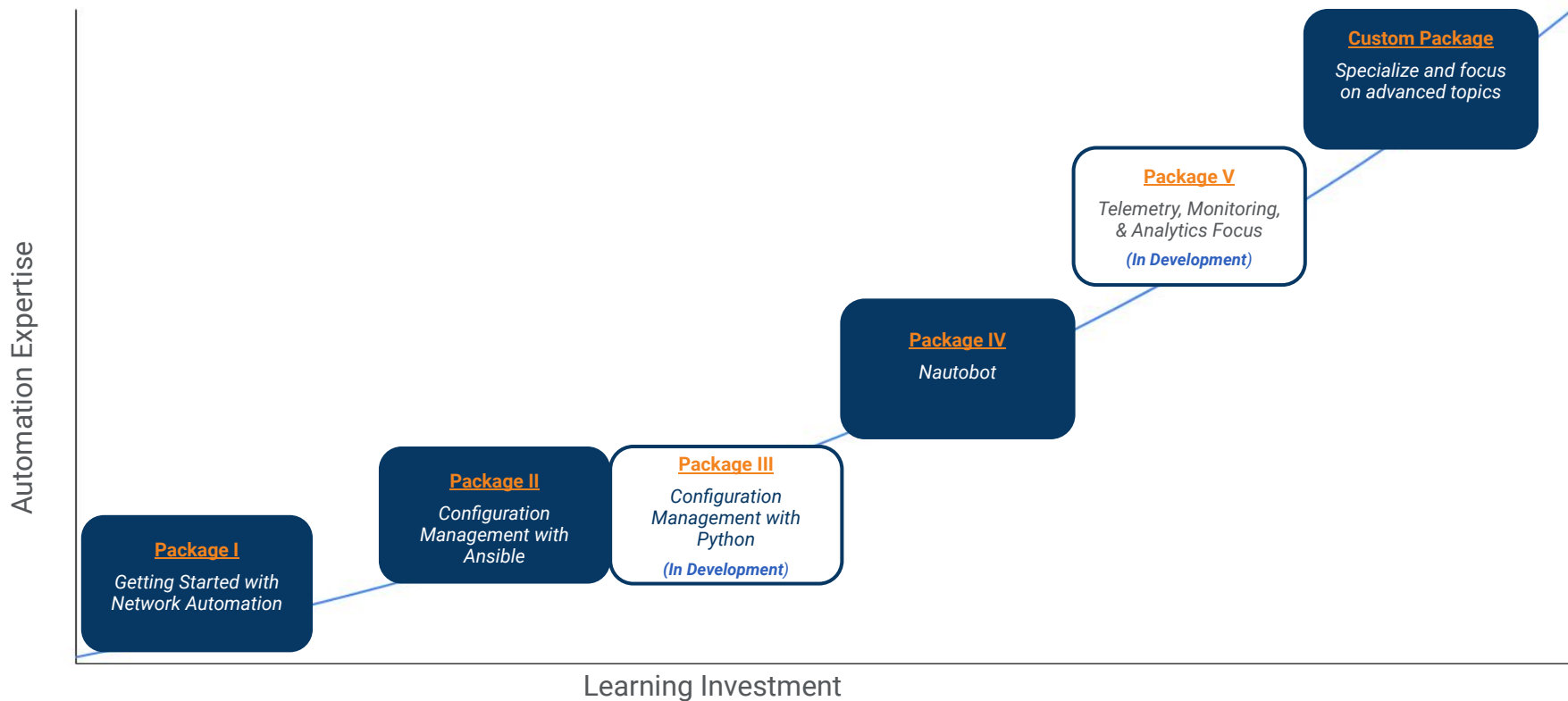


Industry Recognized Thought Leaders

Working with clients across all industries and geographies, we promote a vendor and tool neutral approach, making automation a reality for any network.

>>> The Network Automation Journey: Enablement Packages

Pre-Built Packages from Start to Finish



>>> Housekeeping

Length & Time

Breaks

Course Evaluation: a feedback form link will be sent on the last day. Please take a few minutes to fill this out - **we value your feedback!**



>>> Agenda

- Introduction to Git & Version Control Systems
- Working with a Local Git Repo
 - Demo + Hands-on Labs
- Collaborating with GitHub
 - Demo + Hands-on Labs
- Time Travel with Git
 - Demo
- Collaborating on GitHub Repos with CI
 - Demo + Challenge Lab



Introduction to Git & Version Control Systems

>>> Version Control Systems

Why Version Control?

- Collaboration
- Storing Versions
- Restoring Previous Versions
- Understanding What Happened

Tools: **Git** (most common), Subversion (SVN), Mercurial (HG), CVS, Fossil, Perforce

>>> Git Overview

- Command-line utility created by Linus Torvalds in 2005
- Original purpose was to support multiple collaborators working on Linux Kernel development
- Git is a **distributed** version control system
 - peer-to-peer interaction vs. client-to-server
 - version control = historical change tracking
- Intent
 - Provide version control (core functionality)
 - Foster better code collaboration
 - Reduce mistakes (bugs!)



>>> Git Use Cases

Developers use Git to:

- Work on different versions of code
- See the difference between two or more versions of code
- Review the history of code
- Store code in a shared repository
- Experiment with a new feature without interfering with working code

>>> Git Use Cases - Network Automation

Network Engineers can use Git for version control and collaboration on:

- Network Device Configs
- Playbooks
- Scripts
- Variables Files
- Any file that would benefit from being version controlled!

>>> Collaborative Platforms

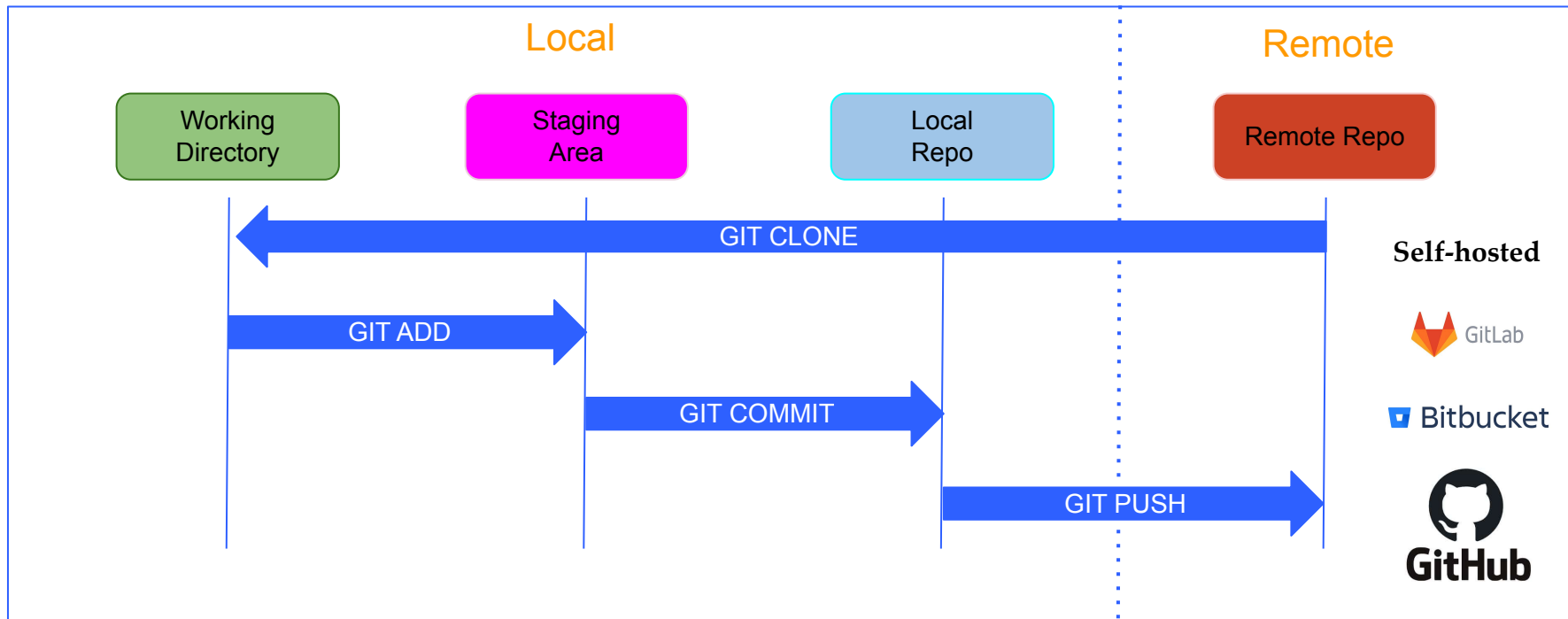
Cloud-based or private on-premises storage for your code.

Common features:

- Web UI
- Viewing code differences between versions
- Merging together code from different feature branches
- Code review capabilities
- Notifications, Discussion Boards
- Support multiple version control systems
- Issue and Project tracking



>>> Git Architecture

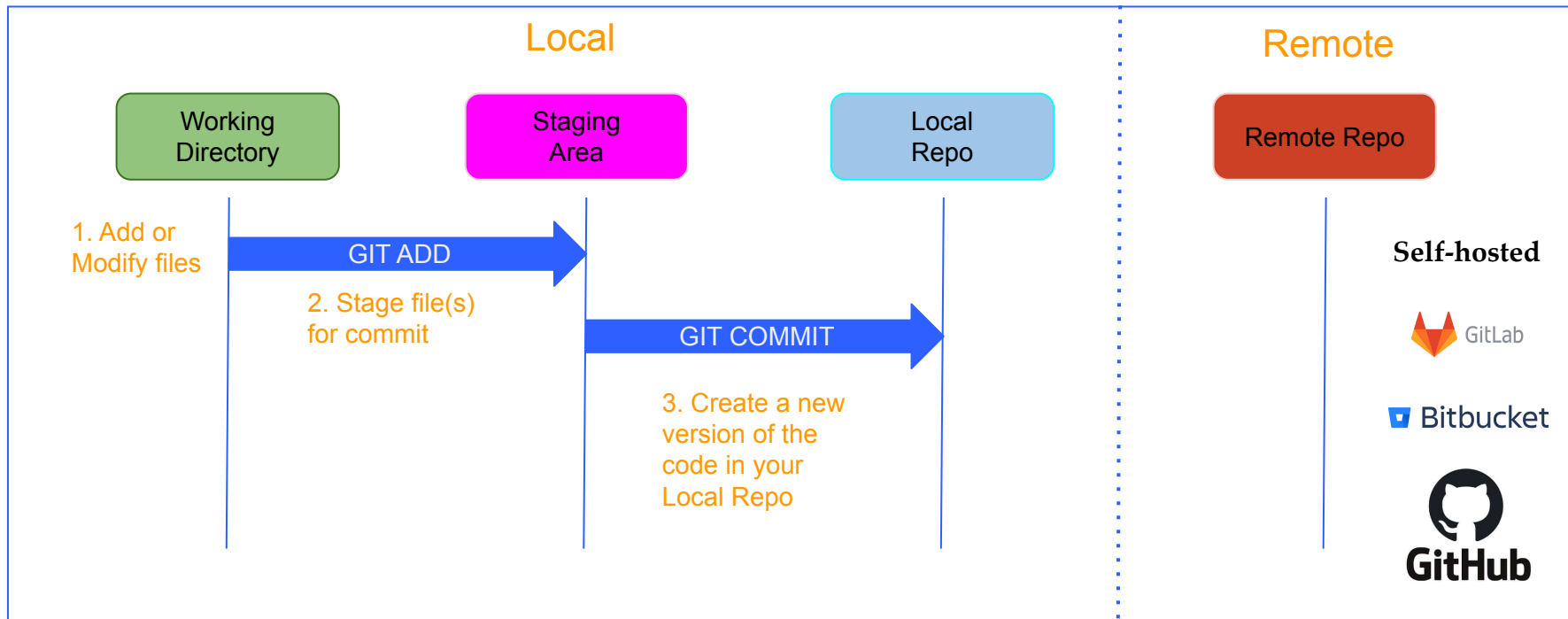




>>> Agenda

- Introduction to Git & Version Control Systems
- **Working with a Local Git Repo**
 - Demo + Hands-on Labs
- Collaborating with GitHub
 - Demo + Hands-on Labs
- Time Travel with Git
 - Demo
- Collaborating on GitHub Repos with CI
 - Demo + Challenge Lab

>>> Git Workflow Overview



>>> Use Cases

- Add VLAN to a config file that is versioned in a Git repository
- Update YAML files
- Update Ansible playbooks
- Update scripts
- And more!

>>> Initialize (Create) a Project

Create a local directory and initialize the repo:

```
$ mkdir git-demo
```

```
$ cd git-demo
```

```
$ git init
```

```
Initialized empty Git repository in /Users/ntc/git-demo/.git/
```

>>> Initialize (Create) a Project

```
git init vlan-configs # Sample git repo name
```

Git init - Creates an empty Git repository or re-initializes an existing one

```
# Creates (if necessary) a repository-named directory  
that contains a new .git subdirectory
```

```
ls -al vlan-configs # shows ., .., & .git directories
```

```
# Changing to vlan-configs may show the git prompt
```

```
cd vlan-configs
```

```
vlan-configs (master #)
```

```
# This branch can be renamed to main
```

```
git branch -m main
```

```
vlan-configs (main #)
```

>>> Create / Modify Files

Create and modify files in the Working Directory.

```
$ cat vlan_config.txt
```

```
vlan 10  
    name Management
```

```
vlan 20  
    name Voice
```

```
vlan 30  
    name Web
```



Add VLAN

```
$ cat vlan_config.txt
```

```
vlan 10  
    name Management
```

```
vlan 20  
    name Voice
```

```
vlan 30  
    name Web
```

```
vlan 40  
    name App
```

>>> Quiz Question

What is likely the most commonly used git command?



>>> Quiz Answer

What is likely the most commonly used git command?

`git status` is a read-only command so it is safe to use any time to help you learn your current situation.



>>> Git Status - use frequently!

Shows files that have been staged for commit as well as untracked files.

Untracked means the files have never been added to staging or committed.

```
$ git status  
On branch main
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
vlan_config.txt
```

```
nothing added to commit but untracked files present (use "git  
add" to track)
```

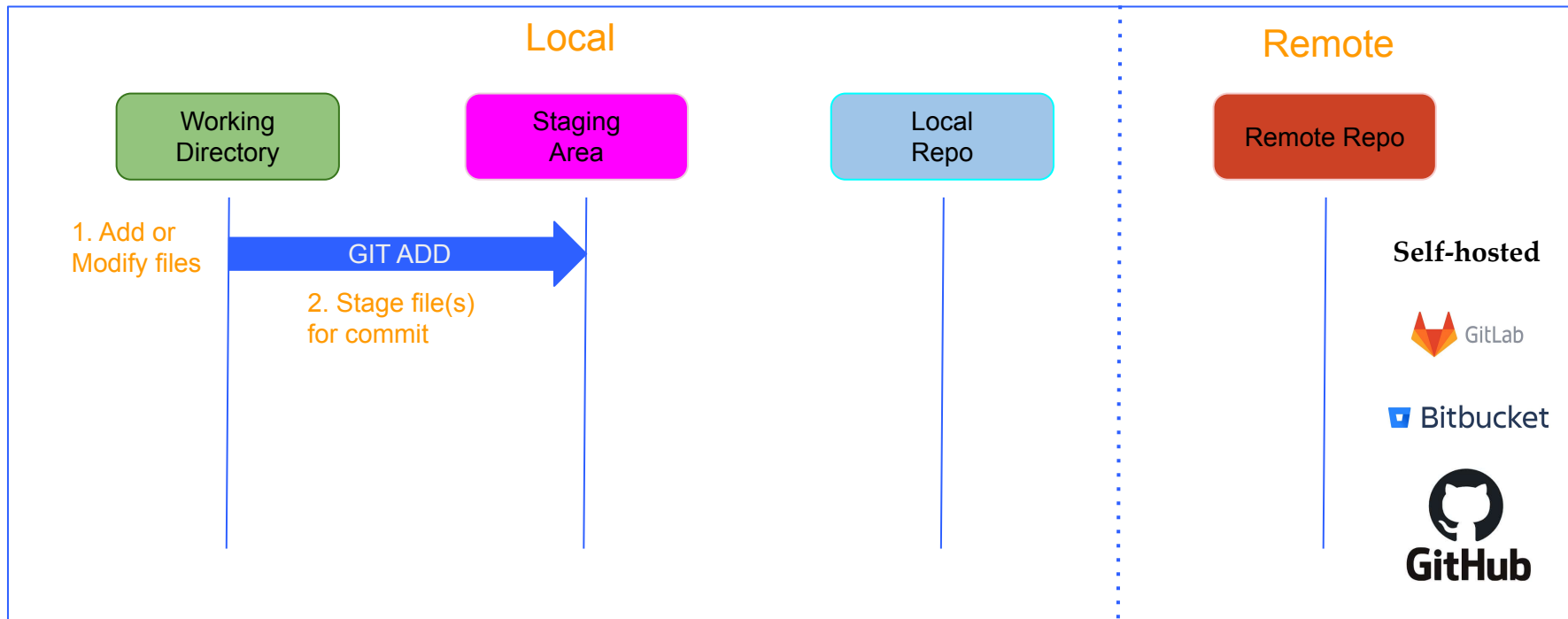
>>> Legacy branch naming convention

Originally the term used for the main branch was 'master'.

The GitHub default changed to 'main' in October 2020.

Should you encounter the former term, when in that branch, you can rename it to the newer one with the command 'git branch -m main'.

>>> Git Workflow Overview



>>> Git Add

Create/Modify files and then execute “git add [filename]”:

```
$ git add vlan_config.txt
```

You can also use "git add ." to add **all** files to the staging area

>>> Git Status

Changes to be committed indicates file has been added to Staging

```
$ git status
```

```
On branch main
```

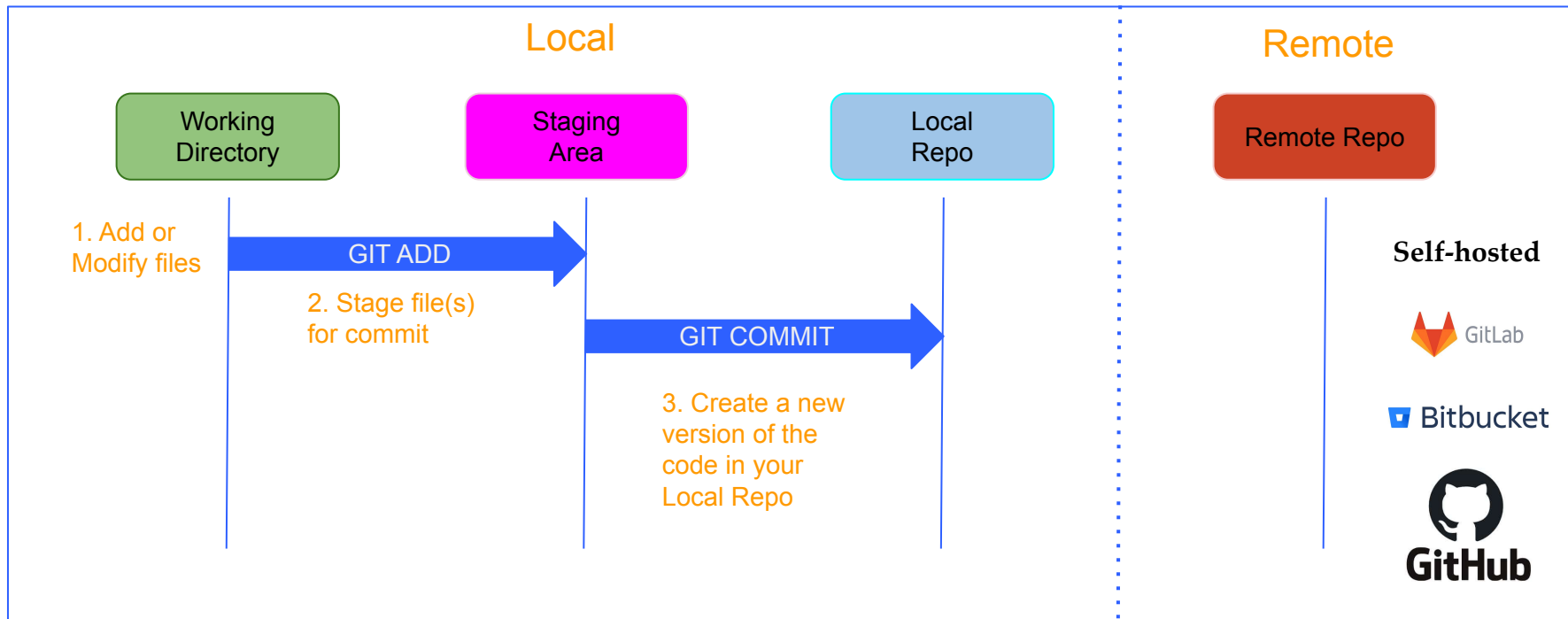
```
No commits yet
```

```
Changes to be committed:
```

```
    (use "git rm --cached <file>..." to unstage)
```

```
new file:   vlan_config.txt
```


>>> Git Workflow Overview



>>> Git Config

When creating a commit and prior to initiating a push, user information must be specified.

```
$ git config --global user.name "John Smith"
$ git config --global user.email john\_smith@networktocode.com
$ git config --list --global # to see current global settings
$ git config --list # to see current local settings
```

This is a one-time configuration that must be done prior to making any commits.

If you forget to configure, git reminds you and provides the commands too!

>>> Git Commit

Creates a snapshot in time in the Local Repo

```
$ git commit -m "first commit"
[main (root-commit) 379327e] first commit
1 file changed, 1 insertion(+)
create mode 100644 vlan_config.txt
```

>>> Git Log

Shows you the Commit History and associated Commit IDs

Commit ID

```
$ git log
```

```
commit 379327e7c9a377156b0a481b6eb92c1918fefaca (HEAD -> main)
```

```
Author: Matt Mullen <matt.mullen@networktocode.com>
```

```
Date:   Wed Jul 8 14:23:20 2020 -0400
```

```
    first commit
```

>>> Show commit logs with `git log --oneline`

```
c78a6ad34 (HEAD -> develop, origin/develop, origin/HEAD) Update release date, version.  
8f669461c Add release-note for #2095  
3a8e0a0c7 Documenting Redis TLS and adjusting healthcheck. (#2097)  
6a51b2041 Add release-notes for #2073, #2080, #2143  
c4d01dbff Add local option to runjob (#2073)  
2f0cf37f5 Update job view to show the class path and add data as option to runjob (#2080)  
453ec9e46 Add custom interval to scheduled job view (#2143)  
709613359 add release note for #2114
```

>>> What do the various forms of HEAD point to?

```
c78a6ad34 (HEAD -> develop, origin/develop, origin/HEAD) Update release date, version.  
8f669461c Add release-note for #2095  
3a8e0a0c7 Documenting Redis TLS and adjusting healthcheck. (#2097)  
6a51b2041 Add release-notes for #2073, #2080, #2143  
c4d01dbff Add local option to runjob (#2073)  
2f0cf37f5 Update job view to show the class path and add data as option to runjob (#2080)  
453ec9e46 Add custom interval to scheduled job view (#2143)  
709613359 add release note for #2114
```

HEAD points to c78a6ad34
HEAD~0 points to c78a6ad34
HEAD~1 points to 8f669461c
HEAD~2 points to 3a8e0a0c7
HEAD~3 points to 6a51b2041
etc.

>>> Viewing Local Diffs

View the difference between your local working directory and latest commit:

```
$ git diff
diff --git a/vlan_config.txt b/vlan_config.txt
index ed40a9b..e69d684 100644
--- a/vlan_config.txt
+++ b/vlan_config.txt
@@ -7,4 +7,6 @@ vlan 20
     vlan 30
         name Web
+vlan 40
+  name App
```

>>> Why use `git mv` and `git rm` versus manual `mv` and `rm`?

- `git mv` and `git rm`:
 - Directly change both the working directory and index
- `mv` and `rm`:
 - Changes only the working directory
 - Must manually then do `git add <filename>` to move to index
- Bottom line:
 - Matter of convenience
 - Less likely to forget to afterwards manually `git add <filename>`



>>> Demo - Working with a Local Repo



>>> Lab 1



>>> Agenda

- Introduction to Git & Version Control Systems
- Working with a Local Git Repo
 - Demo + Hands-on Labs
- **Collaborating with GitHub**
 - Demo + Hands-on Labs
- Time Travel with Git
 - Demo
- Collaborating on GitHub Repos with CI
 - Demo + Challenge Lab

>>> Remote Repository (“repo”)

- Cloud-based or private on-premises storage for your code
- Common features:
 - Web UI
 - Viewing code differences between versions
 - Merging together code from different feature branches
 - Code review capabilities
 - Notifications



Self-hosted

>>> Authentication with GitHub Today - Method 1 of 2

To authenticate over HTTPS for git operations:

- You need to create a PAT (Personal Access Token) which serves as a **scoped temporary password**.
- [Since 2021](#), GitHub is no longer accepting account passwords when authenticating Git operations.

Create Personal Access Token on GitHub:

- From your GitHub account, go to Settings, then Developer Settings, then Personal Access Token.
- Click on Generate New Token. Complete the form and click Generate token.
- Make sure to copy the generated PAT to somewhere safe.
- It will be of a format similar to `ghp_sFhFsSHhTzMDreGRLjmks4Tzuzgthdvfsrta`

You can now use this PAT as a password when authenticating via git commands on the CLI.

Note: based on your operating system, you have a few extra instructions in the extra slides at the end of this presentation.

>>> Authentication with GitHub Today - Method 2 of 2

SSH keys are recommended for general usage on a daily basis (for automating logins, single sign-on, and for authenticating hosts).

- You will need the OpenSSH client installed on your machine (default on Linux distributions or MacOS, installable on Windows 10 or later).
- **ssh-keygen** is a tool for creating new SSH authentication key pairs. You can optionally set a passphrase on your keys so that only you can unlock them.

```
$ ssh-keygen -t rsa -b 4096 # Generate keys by type & bit length, accepting defaults for  
passphrase and the output_keyfile file name where keys are saved
```

- Copy the ssh **public** key to your clipboard. Never let anyone see your **private** key!
- In the upper righthand corner of any GitHub web page, click your profile photo, then Settings, then SSH and GPG keys, then New or Add SSH key, then add a descriptive Title, paste your ssh key into the Key field, and then Add SSH key.
- For more details, check out the GitHub [documentation page](#).

Note: for the NTC labs, so you don't accidentally leave behind your actual SSH keys, we recommend using temporary Personal Access Tokens (PATs) instead.

>>> Pushing to Remote - Prerequisites

1. Create the remote repo
2. Obtain the URL for the remote repo
3. Configure your git profile on your local machine using “git config”
4. Configure the URL to the remote repo using “git remote”

>>> Creating a Repo - GitHub Example

Quick setup — if you've done this kind of thing before

Set up in Desktop or **HTTPS** SSH `https://github.com/networktocode-llc/git-demo.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a `README`, `LICENSE`, and `.gitignore`.

...or create a new repository on the command line

```
echo "# git-demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/networktocode-llc/git-demo.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/networktocode-llc/git-demo.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

💡 ProTip! Use the URL for this page when adding GitHub as a remote.

Callouts:

- URL**: Points to the repository URL `https://github.com/networktocode-llc/git-demo.git`.
- New Repo Commands**: Points to the command list for creating a new repository.
- Existing Repo Commands**: Points to the command list for pushing an existing repository.

>>> Git Remote

Remote always called
"origin" by convention

Sets the URL to the remote repo

```
$ git remote add origin https://github.com/ntc-training/git-demo.git
```

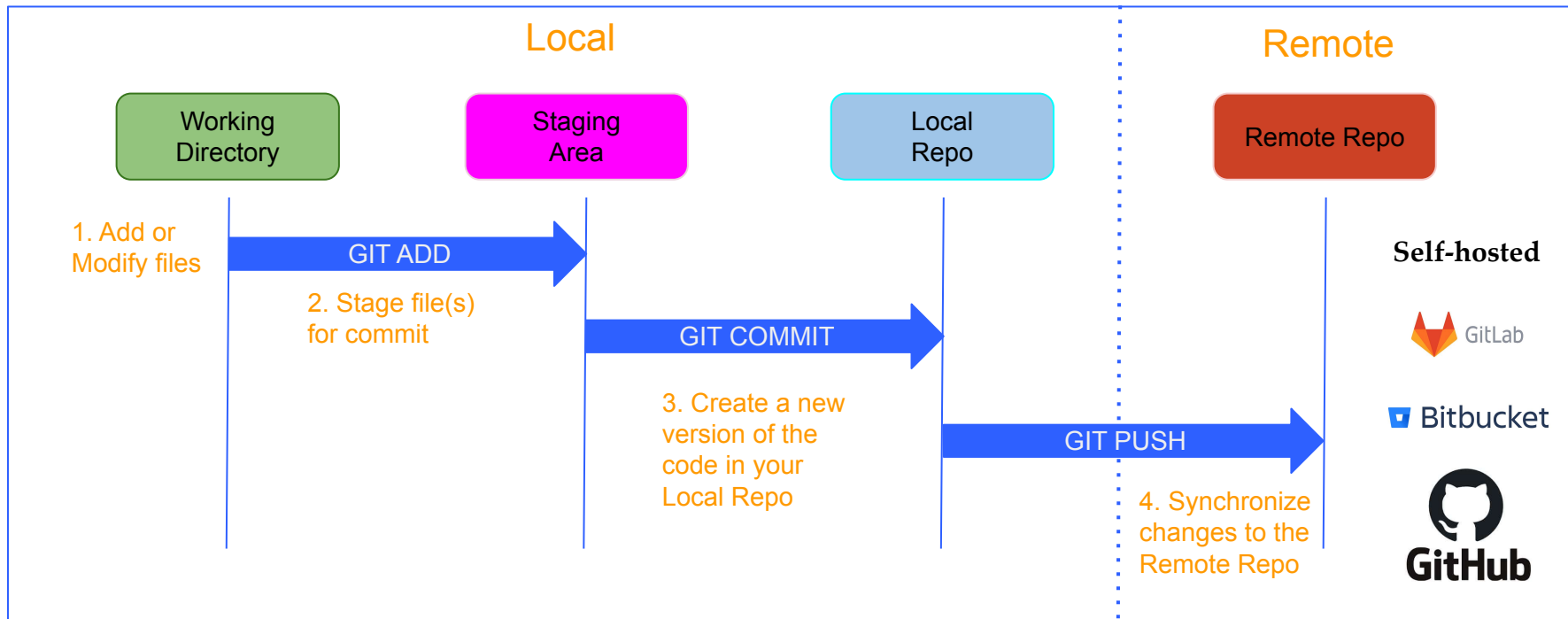
View the URL to the remote repo

```
$ git remote -v
```

```
origin      https://github.com/ntc-training/git-demo.git (fetch)
```

```
origin      https://github.com/ntc-training/git-demo.git (push)
```

>>> Git Push



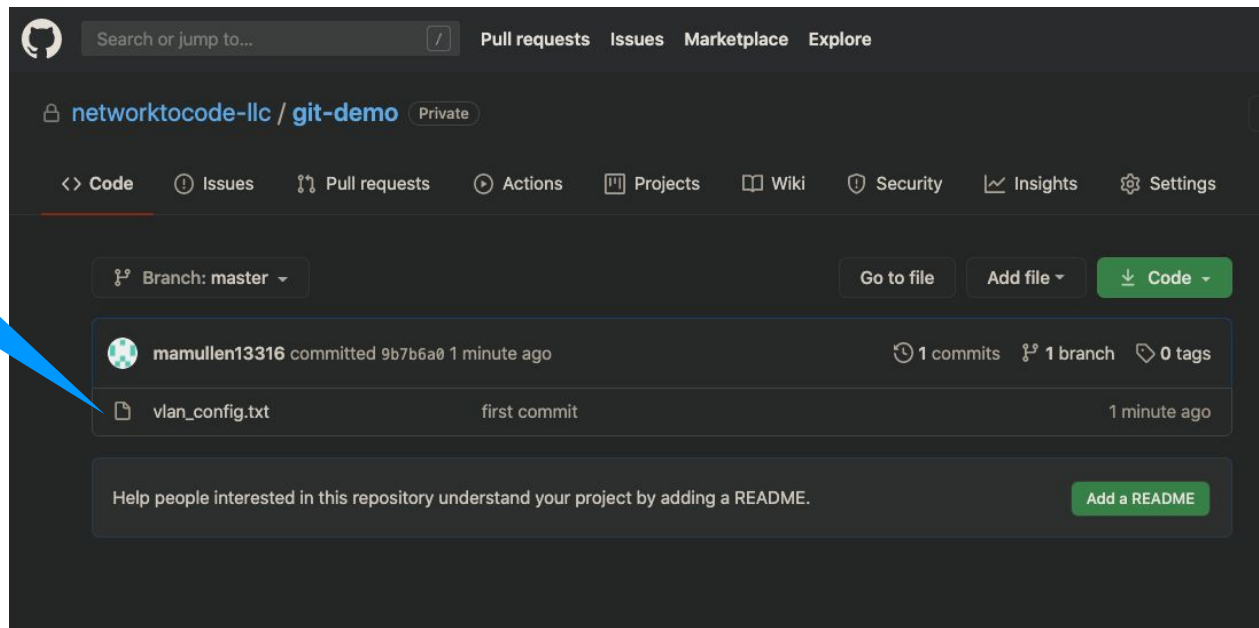
Copy the files from the Local Repo to the Remote

```
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 228 bytes | 228.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ntc-training/git-demo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

main = branch
name.

-u is short for
--set-upstream and is only
needed on the first push.

vlan_config.txt has
been pushed and now in
the repo



>>> More Changes - Rinse and Repeat

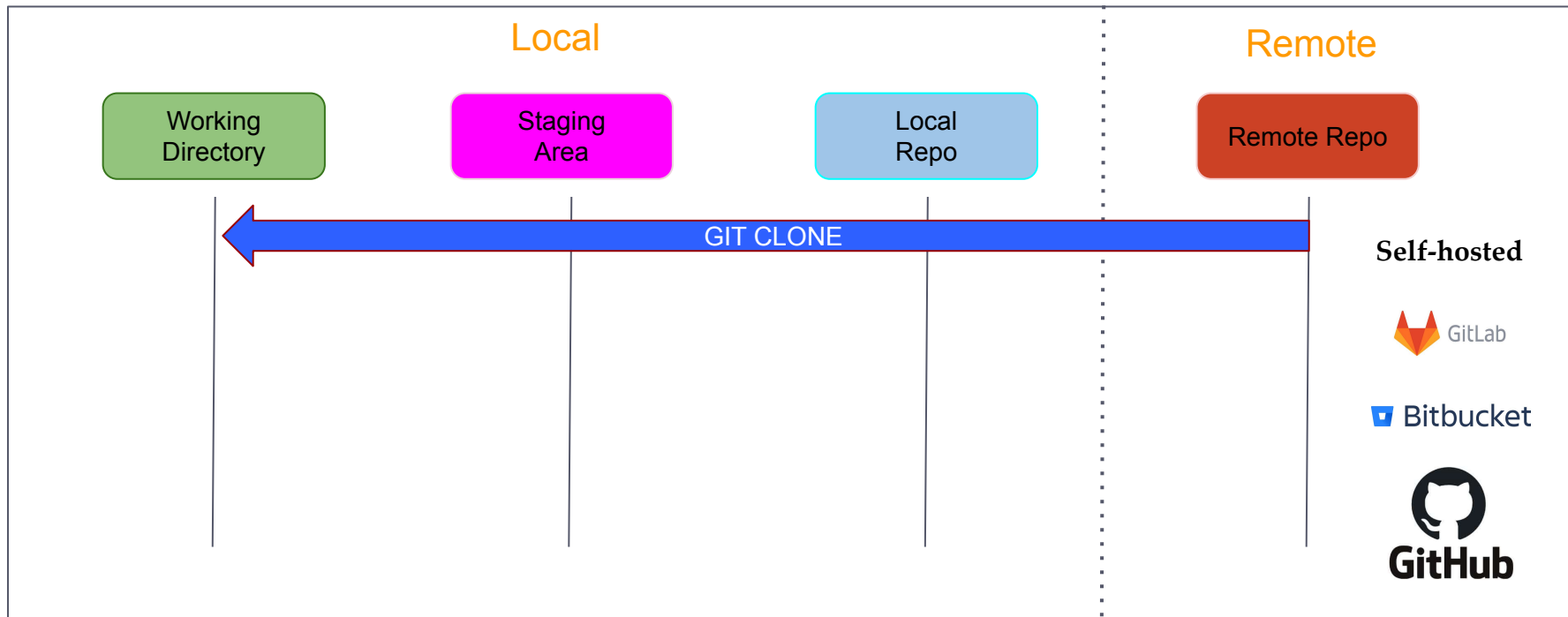
1. Make changes/additions/etc in the Working Directory
2. `git add .`
3. `git commit -m "commit message"`
4. `git push`

"." means all files and directories under the current directory

>>> Cloning

- One-time operation to pull down a copy of a remote repo into your local working directory
- The project retains the same remote URL as the original project
- Accomplished using the `git clone [url]` command

>>> Git Clone



>>> Git Clone Example

```
$ git clone https://github.com/ntc-training/git-demo.git git-demo-clone
```

```
Cloning into 'git-demo'...
```

```
remote: Enumerating objects: 3, done.
```

```
remote: Counting objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
```

```
Receiving objects: 100% (3/3), done.
```

```
$ ls
```

```
git-demo
```

```
$ cd git-demo
```

```
$ ls
```

```
test.txt
```

Optionally specify the directory we are cloning into. If left out, it will be the same as the repo name.

>>> Git Clone

- One-time operation
- Copy from remote
- **Same remote url as original**
- Syntax:

```
git clone [url]
```

```
git clone [url] [dir]
```

>>> Git Branch

A parallel copy of the project in which you can add new features and functionality non-disruptively.

To create a branch:

```
$ git branch mybranch
```

And then work on it:

```
$ git checkout mybranch
```

To create a new branch and work on it:

```
$ git checkout -b mybranch
```

OR (new syntax)

```
$ git switch -c mybranch
```

>>> Changing Branches

To change branches:

```
$ git checkout mybranch
```

OR

```
$ git switch mybranch
```

New in git
version 2.23.0

You are now working in the new branch with an exact copy of the files as they existed in the original branch (main)

>>> Viewing Branches

To view branches:

```
$ git branch
```

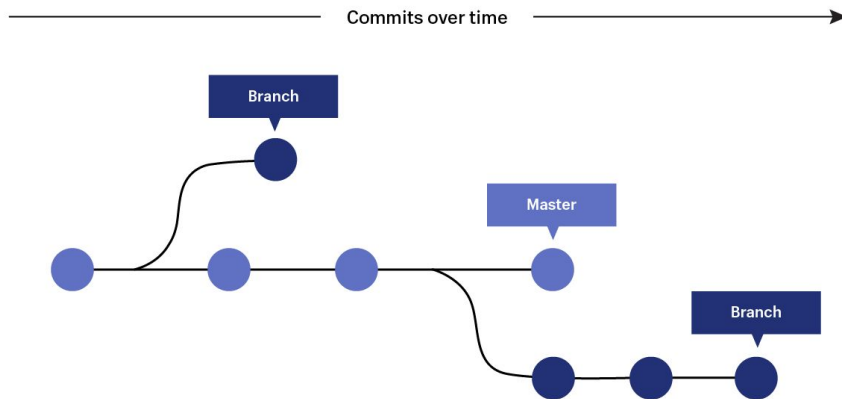
```
main
```

```
* mybranch
```

The current branch will have an asterisk (*) next to it

>>> Git Branches

- **Repository (repo)** - A project or collection of work being managed by git
- **Branch** - A particular series of changes to content within the repo
 - Branches can diverge and merge, like paths through a forest
 - The default branch is typically named main by convention
 - Branches may be temporary or permanent



>>> Git Branches

- Commits allow us to move forward and backward in time
- Branches **exist in parallel** to one another
 - Each branch has its own series of commits
- Branches are “**checked out**” to the current workspace
- Disruptive development can be confined to a particular branch without sacrificing the stability of another
 - Example: **main** versus **develop** branches
 - Example: **bugfix** and **feature-add** branches
- Keep branches focused on one thing and as short-lived as possible



Demo - Git Branching and GitHub



>>> Lab 2

>>> Forking on GitHub to collaborate with others

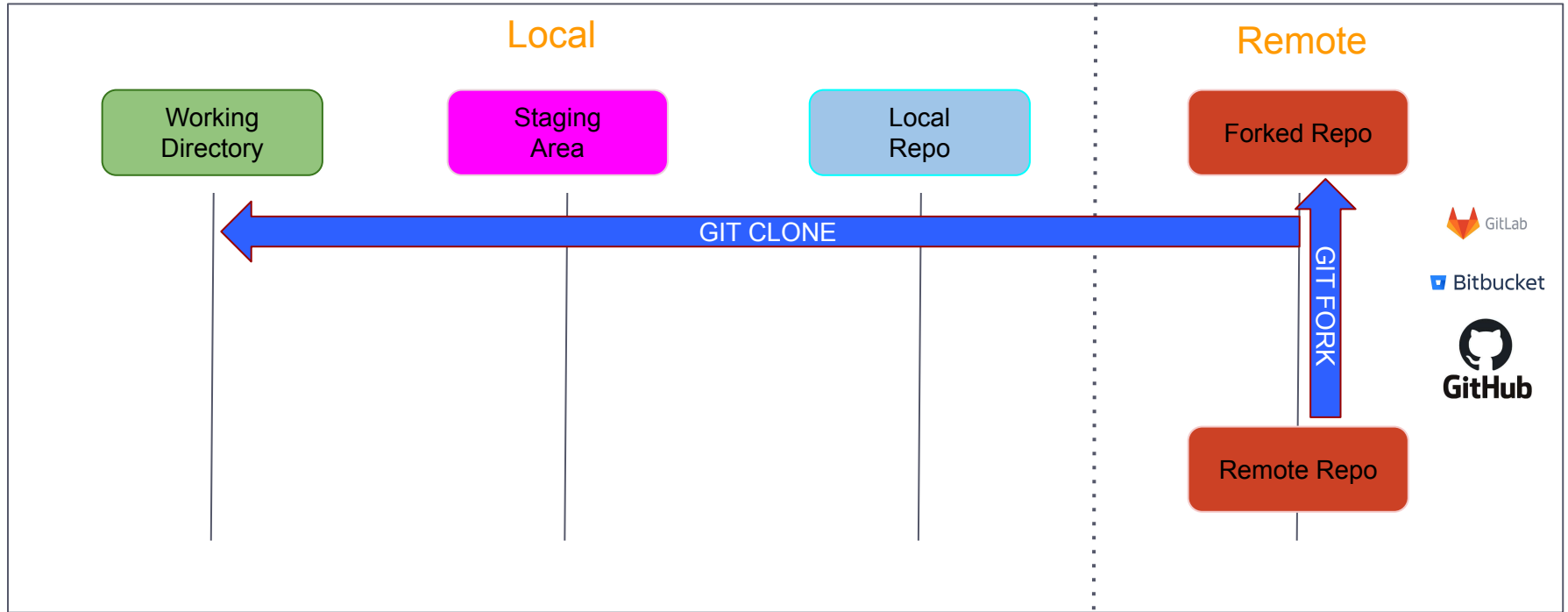
- Creates a copy of the remote repo on the remote repository server
- The copy is given a brand new name and URL (which aligns to your GitHub / BitBucket account)
- It is accomplished using the GUI of the remote repo, rather than a git command
- Clone the forked copy to work on it locally

>>> GitHub Fork

- One-time operation
- Copy from remote
- **Different remote url than original**
- Once you fork, you can clone from the new url
- Syntax:

```
git remote add origin  
https://github.com/example/example.git
```

>>> Git Fork, then Git Clone



>>> Git Fork Example

The screenshot shows the GitHub interface for the repository 'networktocode-llc / git-demo'. The repository is marked as 'Private'. In the top right corner, there are buttons for 'Watch' (7), 'Star' (0), and 'Fork' (0). A red arrow points to the 'Fork' button. A modal dialog titled 'Fork git-demo' is open in the center. The modal asks 'Where should we fork git-demo?' and lists two options: 'mamullen13316' and 'network-automation'. Below this, it says 'Can't find what you're looking for?' and 'You don't have permission to fork to these organizations:', followed by a list containing 'networktocode'.

>>> Quiz Question

When should you **fork** a repository versus **clone** it with `git`?



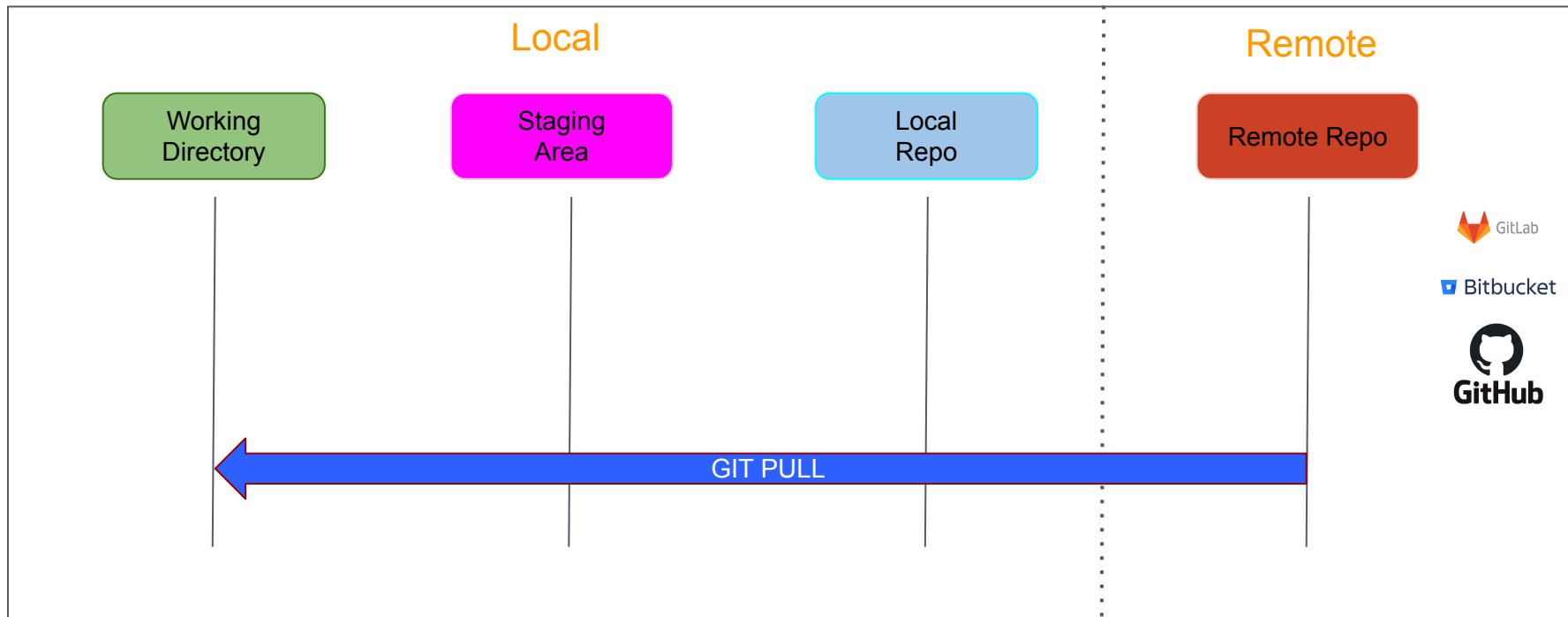
Clone Use-Cases

- You just want to copy, use, or explore the project
- You have write access to the repo and therefore, can push back up directly

Fork Use-Cases

- You want to contribute to a project you don't have write access to
- Standard approach is to fork'n'pull vs. clone/push

>>> Git Pull



>>> Git Pull

First someone else makes a change and pushes to the remote.

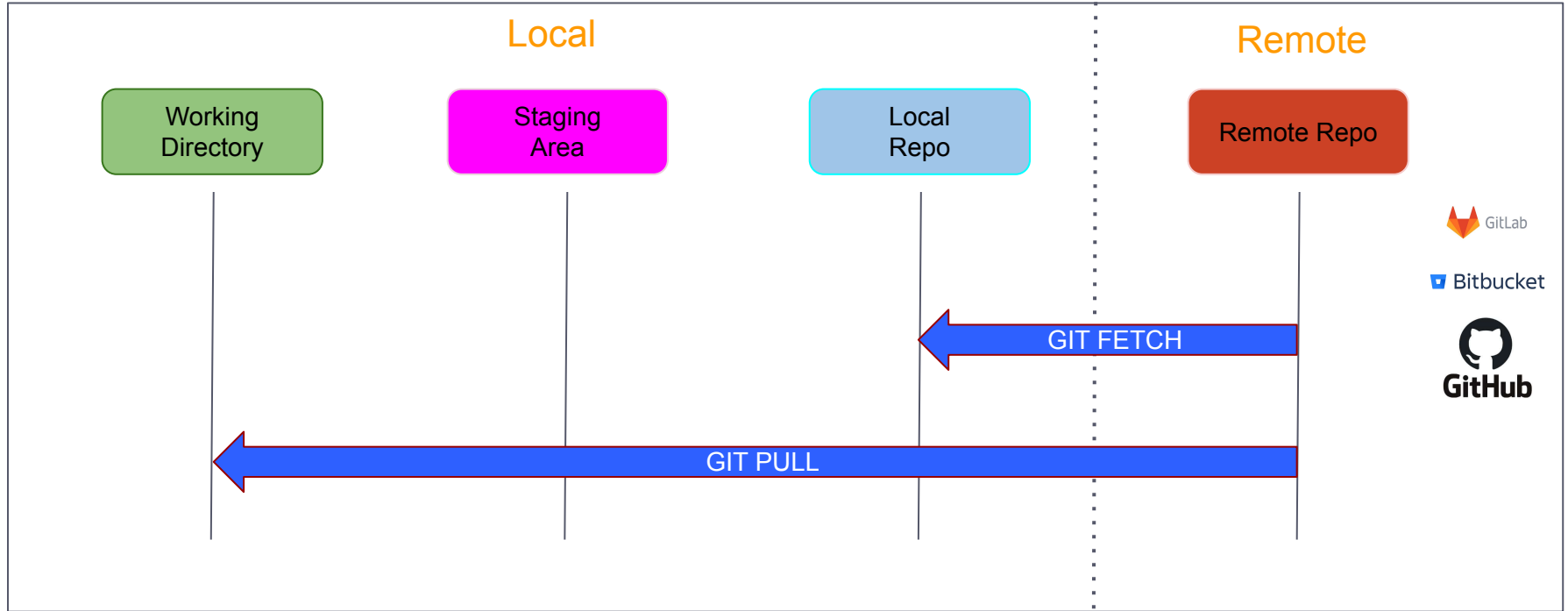
```
$ echo "created another file" > test2.txt
$ ls
test.txt test2.txt
$ git add .
$ git commit -m "file added by some user"
```

For this task we are simulating another user making a change by making a change in the cloned copy of the repo.

Pull the contents of the remote repo into your working directory:

```
$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), 574 bytes | 191.00 KiB/s, done.
From https://github.com/ntc-training/git-demo
    2201f74..84d5247  main      -> origin/main
Updating 2201f74..84d5247
Fast-forward
 test2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test2.txt
```

>>> Git Fetch



>>> Git Fetch

User makes a change in the remote repo

```
$ echo "changed by another user" >> test.txt  
$ git add .  
$ git commit -m "text.txt changed by another user"
```

In this task we are again simulating a user making a change by making the change in the cloned repo and pushing it to the remote.

>>> Git Fetch

Pull down changes from the remote to your Local Repo:

```
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 260 bytes | 130.00 KiB/s, done.
From https://github.com/ntc-training/git-demo
   369dcea..45239bb  main      -> origin/main
```

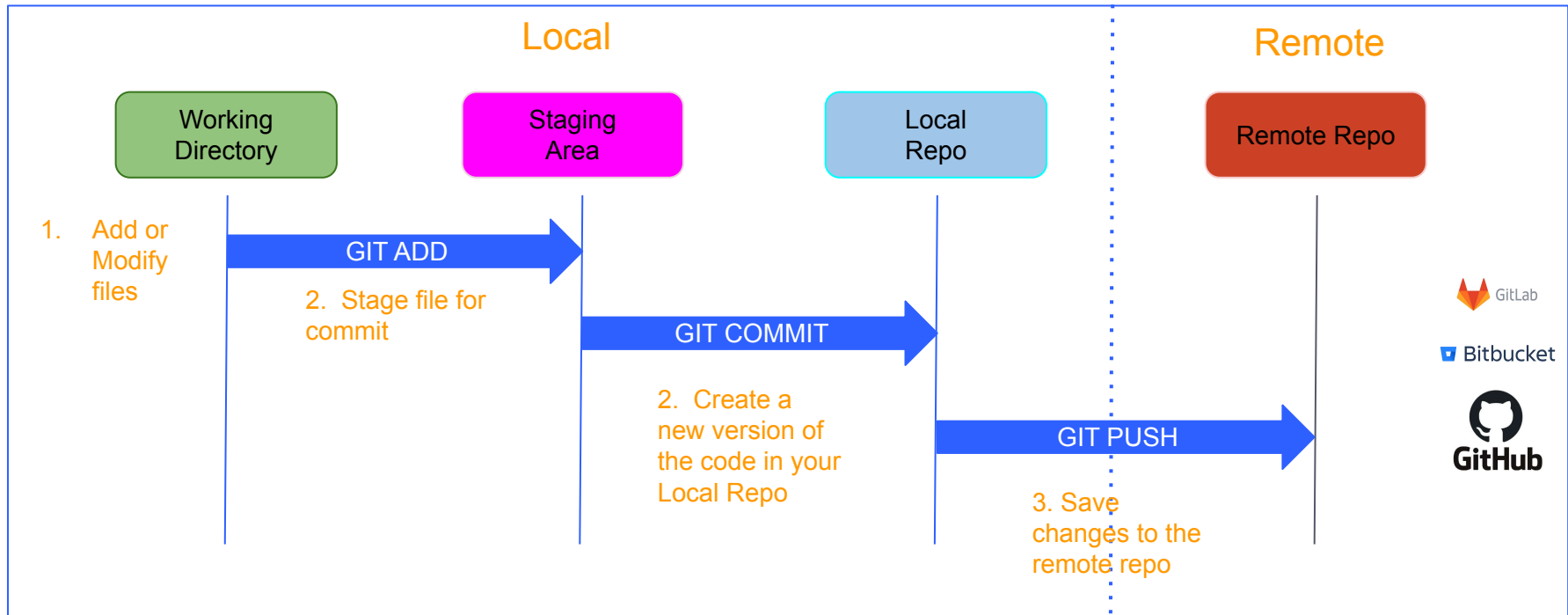
Notice that the changes in the remote are not yet in the file in your working directory, that's because they are in the Local Repo!

>>> Quiz Question

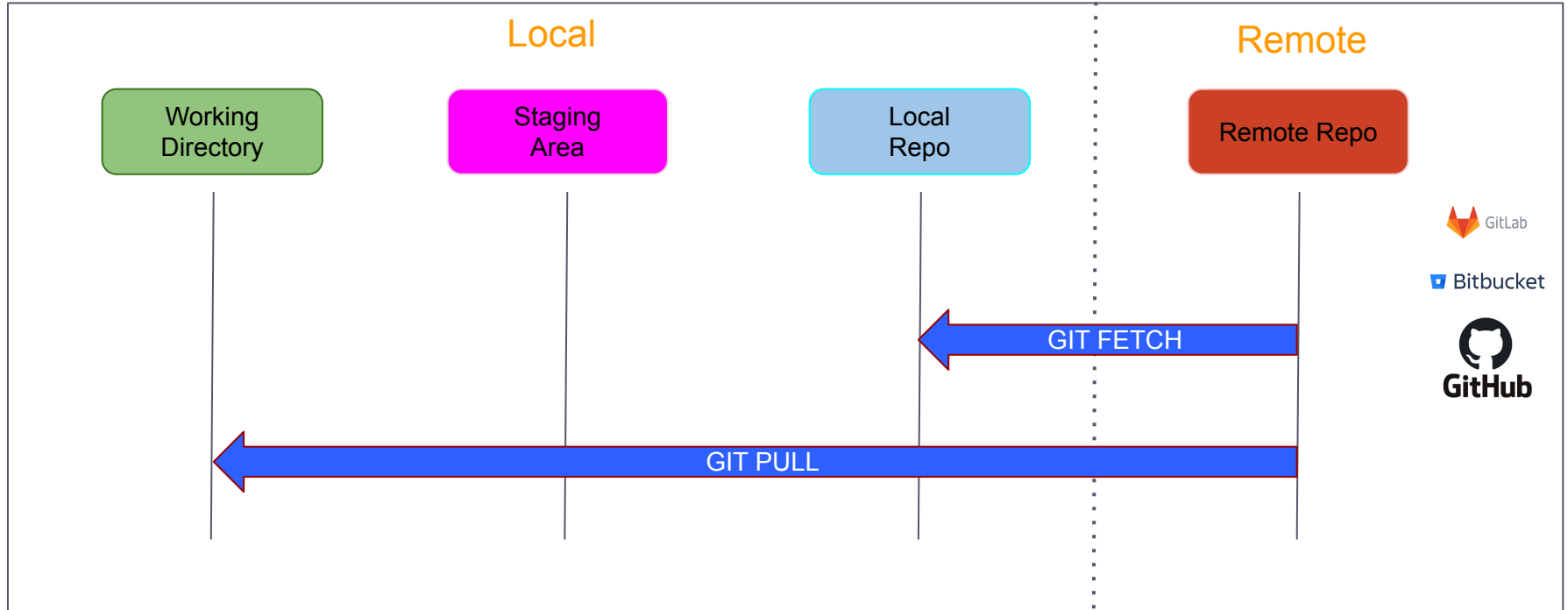
What is the opposite of `git push`?



>>> Opposite of Git Push is NOT Git Pull



>>> Opposite of Git Push is Git Fetch



>>> Comparing the Working Directory with Remote

```
$ git diff origin/main
diff --git a/test.txt b/test.txt
index 9439068..6d0f058 100644
--- a/test.txt
+++ b/test.txt
@@ -1,2 +1,2 @@
    first edit
-changed by another user
+second commit
```

Reference to the
Local Repo
which now has a
copy of the
Remote

Minus sign
means this line
doesn't exist in
our local copy of
the file

Always do a “git fetch” first, otherwise you might be looking at old data in your Local Repo

>>> Merge Conflict

What if we try to pull now?

```
$ git pull origin main
```

```
Updating 69742b6..ba91a8f
```

```
error: Your local changes to the following files  
would be overwritten by merge:
```

```
test.txt
```

```
Please commit your changes or stash them before  
you merge.
```

```
Aborting
```

>>> Merge Conflict

Let's take the advice and commit our changes...

```
$ git add .
```

```
$ git commit -m "third commit"
```

```
[main 1abe23f] third commit
```

```
1 file changed, 1 insertion(+)
```

```
$ git pull
```

```
Auto-merging test.txt
```

```
CONFLICT (content): Merge conflict in test.txt
```

```
Automatic merge failed; fix conflicts and then commit  
the result.
```

>>> Correcting the Conflict

The file in conflict will have markers where the conflict arose...

```
created file
```


```
<<<<<< HEAD
```

```
second commit
```

```
=====
```

```
changed by another user
```

```
>>>>>> ba91a8fed3ca5f89b93a3996d2327f2eda2091fc
```



Commit ID that
created the
conflict

Remove the lines with <<<<<, >>>>>, and =====, keep the contents that are correct, and then save the changes

>>> Commit and Push the Changes

```
$ git add .  
$ git commit -m "resolving merge conflict"  
[main 72a5e56] resolving merge conflict  
$ git push  
Enumerating objects: 10, done.  
Counting objects: 100% (10/10), done.  
Delta compression using up to 12 threads  
Compressing objects: 100% (5/5), done.  
Writing objects: 100% (6/6), 617 bytes | 617.00 KiB/s, done.  
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/ntc-training/git-demo.git  
    ba91a8f..72a5e56  main -> main
```

>>> Make Changes in the New Branch

Create a new file and commit it:

```
$ echo "new file in branch" > test3.txt
$ git add test3.txt
$ git commit -m "test3 added in feature branch"
[mybranch e852896] test3 added in feature branch
 1 file changed, 1 insertion(+)
 create mode 100644 test3.txt
```

>>> Push the Changes

```
$ git push -u origin mybranch
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 323 bytes | 323.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'mybranch' on GitHub by visiting:
remote:   https://github.com/ntc-training/git-demo/pull/new/mybranch
remote:
To https://github.com/ntc-training/git-demo.git
 * [new branch]      mybranch -> mybranch
Branch 'mybranch' set up to track remote branch 'mybranch' from 'origin'.
```

Set-upstream is
required for the
initial push

New branch
created on the
remote

>>> New Branch on GitHub

Search or jump to... / Pull requests Issues Marketplace Explore

networktocode-llc / git-demo Private Watch 7 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

mybranch had recent pushes 6 minutes ago Compare & pull request

Branch: master Go to file Add file Code

Switch branches/tags

Find or create a branch...

Branches Tags

✓ master default

mybranch

View all branches

minutes ago	5 commits 2 branches 0 tags
resolving merge conflict	13 minutes ago
file added from clone	31 minutes ago

understand your project by adding a README. Add a README

About Git demo for training

Releases No releases published Create a new release

Packages No packages published Publish your first package



Creating a Pull Request

The screenshot shows the GitHub interface for the repository `networktocode-llc / git-demo`. The repository is marked as `Private`. The navigation bar includes links for `Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, `Insights`, and `Settings`. The main content area shows the `mybranch` branch, which is 1 commit ahead of `master`. A green button labeled `Compare & pull request` is circled in red. Below this, a commit by `mamullen13316` is shown, with a table of changes:

File	Change	Time
<code>test.txt</code>	resolving merge conflict	14 minutes ago
<code>test2.txt</code>	file added from clone	32 minutes ago
<code>test3.txt</code>	test3 added in feature branch	7 minutes ago

At the bottom, there is a prompt to `Add a README`. The right sidebar contains sections for `About` (Git demo for training), `Releases` (No releases published), and `Packages` (No packages published).



Merge Pull Request

test3 added in feature branch #1

Open mamullen13316 wants to merge 1 commit into master from mybranch

Conversation 0 Commits 1 Checks 0 Files changed 1

mamullen13316 commented 1 minute ago

No description provided.

test3 added in feature branch e852896

Add more commits by pushing to the mybranch branch on networktocode-llc/git-demo.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request You can also open this in GitHub Desktop or view command line instructions.

Write Preview H B I

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close pull request Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

>>> Switch to Main

```
$ git switch main
```

```
Switched to branch 'main'
```

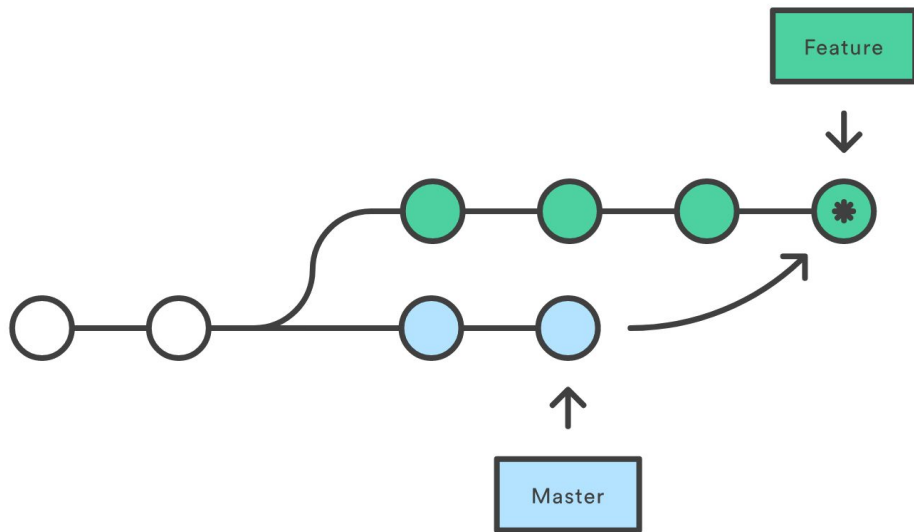
```
Your branch is up to date with 'origin/main'.
```

>>> Pull Remote Changes

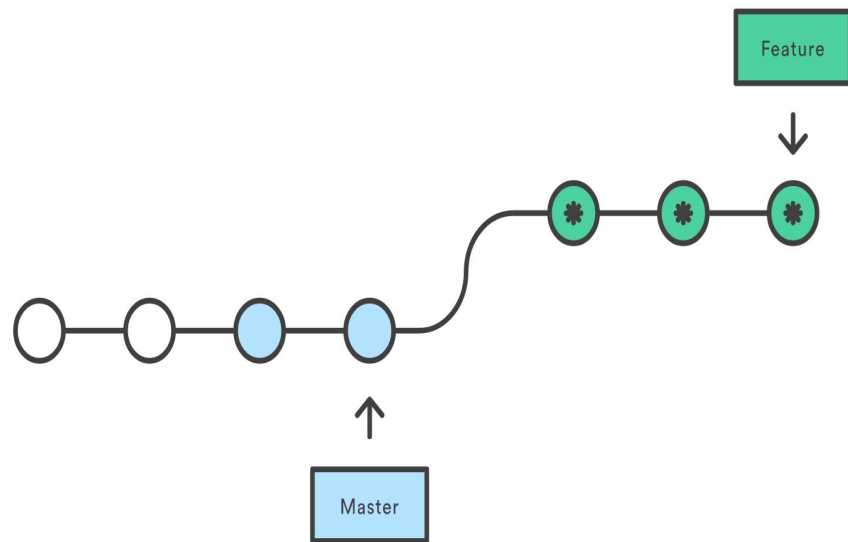
```
$ git pull origin main
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 641 bytes | 641.00 KiB/s, done.
From https://github.com/ntc-training/git-demo
   72a5e56..1f16701  main      -> origin/main
Updating 72a5e56..1f16701
Fast-forward
 test3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test3.txt
```

We have now integrated the changes from the branch into main

>>> Git Merge vs Rebase



* Merge commit



* Brand new commit

>>> Git Merge vs Rebase

- Rebase benefits
 - Streamlines a potentially complex history
 - Avoids merge commit “noise” in busy repos with busy branches
 - Cleans intermediate commits by making them a single commit
- **Never rebase a public or protected branch (such as those *multiple people might work on*)!** You may rewrite project history and destroy teammates changes!!!



Demo Collaborating with the GitHub Fork Model



>>> Lab 3



>>> Agenda

- Introduction to Git & Version Control Systems
- Working with a Local Git Repo
 - Demo + Hands-on Labs
- Collaborating with GitHub
 - Demo + Hands-on Labs
- **Time Travel with Git**
 - Demo
- Collaborating on GitHub Repos with CI
 - Demo + Challenge Lab

>>> Time Travel with `git` COMMAND FLAG HEAD~n (back n commits)

COMMAND	checkout	reset	reset	reset	revert
FLAG		--soft	--mixed (default)	--hard	
Use with repo type	Local	Local	Local	Local	Remote
Commit history	No changes	Alters existing	Alters existing	Alters existing	Adds new commit
Uncommit changes?	No	Yes	Yes	Yes	No
Unstage changes?	No	No	Yes	Yes	No
Delete changes?	No	No	No	Yes	No

>>> Saving work-in-progress with `git stash`

- `git stash` # save staged changes in the stash stack
- `git stash --include-untracked` # save unstaged changes
- `git stash list` # list the stashes
- `git stash show` # show the latest change
- `git stash pop` # put in current working branch & remove from stack
- `git stash clear` # Clear all stashes from the stack
- `git stash branch <branch_name>` # Create new branch from latest stash



Demo - Git Time Travel



>>> Agenda

- Introduction to Git & Version Control Systems
- Working with a Local Git Repo
 - Demo + Hands-on Labs
- Collaborating with GitHub
 - Demo + Hands-on Labs
- Time Travel with Git
 - Demo
- **Collaborating on GitHub Repos with CI**
 - Demo + Challenge Lab

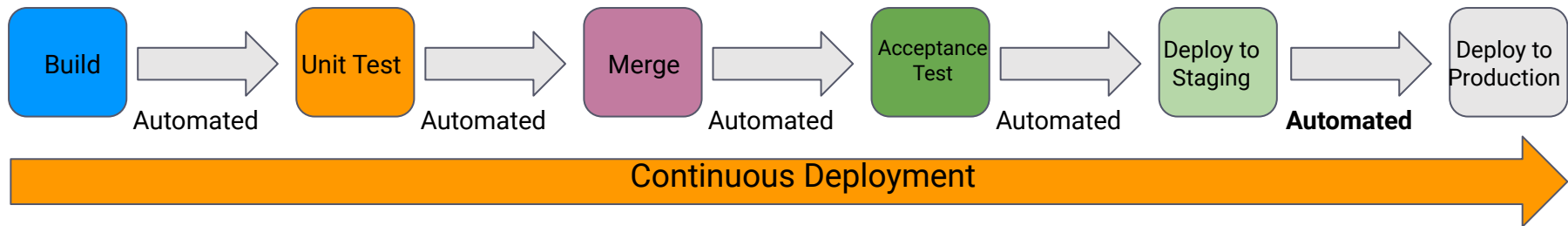
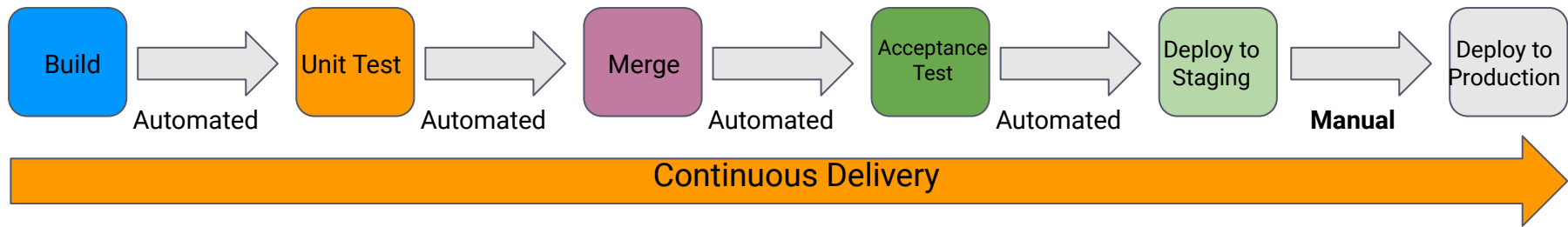
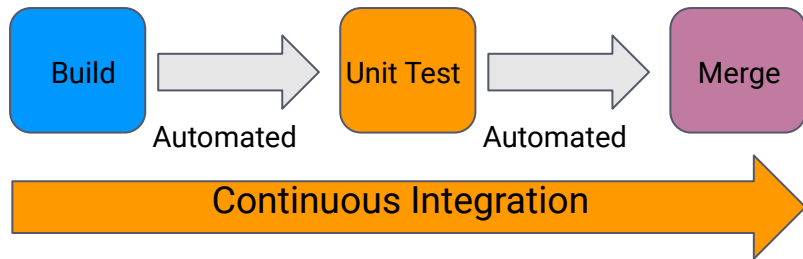
>>> Introduction to CI/CD

- **CI** is the practice of merging code changes frequently and leveraging automated testing - this builds trust in the system and healthier code.
 - “Code” can be anything - Ansible playbooks, data models, device configs etc.
- CI is often paired with **CD** - i.e. Continuous Delivery / Deployment.
 - A pipeline can test but also deploy your “code” to a staging environment, to production, package on PyPi, Ansible collection on Galaxy etc.
- These may be Software-as-a-Service (**SaaS**) providing a cloud-based **continuous integration (CI)** server or **On-Premise**.
 - They may integrate with code hosted on Bitbucket, GitHub, GitLab, etc.
 - Work steps performed (the pipeline) are commonly defined in a YAML file
- Example platforms: Circle-CI, GitHub Actions, Gitlab-CI, Jenkins, Travis, etc.

>>> How to Choose Your CI/CD Tool?

- Already used within your organization
- Can access the code where it is already stored
- Has the necessary features
- Open source vs Commercial & Degree of Support Available
- Wide acceptance in your industry
- Wide acceptance in general

>>> What is CI CD?



>>> Test-Driven Development - TDD

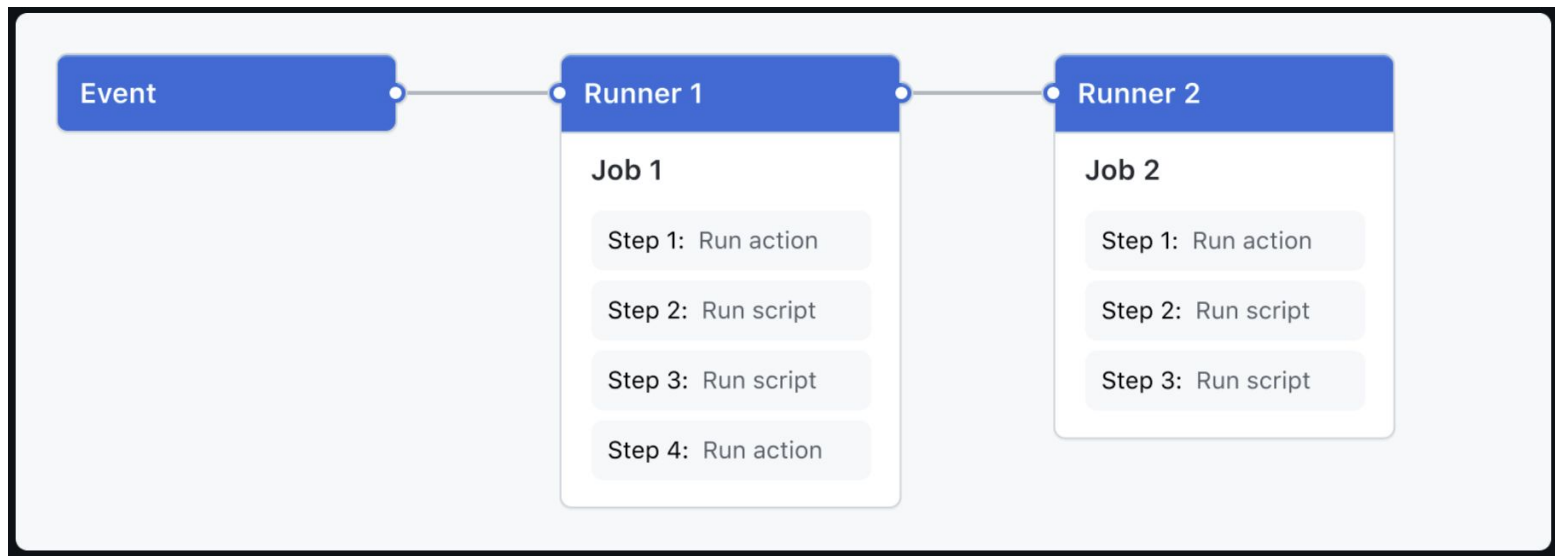
“Test-driven development (TDD) is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is as opposed to software being developed first and test cases created later.”

- https://en.wikipedia.org/wiki/Test-driven_development

Because of the presence of Unit Tests, Acceptance Tests, and other types of testing, TDD may (but doesn't always) integrate in with CI/CD.

>>> Components of GitHub Actions (one example of CI/CD)

You can configure a GitHub Actions workflow to be triggered when an event occurs in your repository, such as a pull request being opened or an issue being created. Your workflow contains one or more jobs which can run in sequential order or in parallel. Each job will run inside its own virtual machine runner, or inside a container, and has one or more steps that either run a script that you define or run an action, which is a reusable extension that can simplify your workflow.



>>> GitHub Actions Marketplace

<https://github.com/marketplace?type=actions>

Actions

Categories

API management

Chat

Code quality

Code review

Continuous integration

Dependency management

Deployment

IDEs

Learning

Localization

Mobile

Monitoring

Project management

Publishing

Recently added

Security

Support

Testing

Utilities

Actions

An entirely new way to automate your development workflow.

14087 results filtered by Actions

Actions

First interaction
By actions
Greet new contributors when they create their first issue or open their first pull request
☆ 140 stars

Close Stale Issues
By actions
Close issues and pull requests with no recent activity
☆ 587 stars

Download a Build Artifact
By actions
Download a build artifact that was previously uploaded in the workflow by the upload-artifact action
☆ 539 stars

Setup .NET Core SDK
By actions
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository
☆ 501 stars

Upload a Build Artifact
By actions
Upload a build artifact that can be used by subsequent workflow steps
☆ 1.6k stars

Setup Node.js environment
By actions
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH
☆ 2.2k stars

Setup Go environment
By actions
Setup a Go environment and add it to the PATH
☆ 809 stars

Setup Java JDK
By actions
Set up a specific version of the Java JDK and add the command-line tools to the PATH
☆ 784 stars

Cache
By actions
Cache artifacts like dependencies and build outputs to improve workflow execution time
☆ 3k stars

action-git-diff-suggestions
By getsentry
This GitHub Action will take the current git changes and apply them as GitHub code review suggestions
☆ 11 stars



Demo - CI/CD with GitHub Actions

>>> Recap

- Git is great for collaboration and version control
- It has a learning curve
- CLI can make you more efficient
- VSCode extensions can improve productivity, e.g. GitLens

>>> The Challenge

- **Lab 04 is a Challenge Lab!**
- You have to fix all the errors in the <https://github.com/ntc-training/codelint> repository
 - There are Python syntax and formatting errors
 - There are YAML syntax errors
 - There are Ansible specific errors
- Submit a Pull Request with your fixes and see that the CI Build is Green (Passed) in your PR.





>>>network.toCode()

Thank You
NetworkToCode.com



>>> Extra Slides

>>> Useful Git and GitHub Resources

- Git Pro eBook: <https://git-scm.com/book/en/v2>
- GitHub Account Setup and Configuration:
<https://git-scm.com/book/en/v2/GitHub-Account-Setup-and-Configuration>
- GitHub's Git Cheat Sheet: <https://education.github.com/git-cheat-sheet-education.pdf>
- Git External Links: <https://git-scm.com/doc/ext>
- Dang it Git: <https://dangitgit.com/en>
- A visual guide to Git internals:
<https://www.freecodecamp.org/news/git-internals-objects-branches-create-repo/>
- Git Commit Comment Humor - <https://xkcd.com/1296/>

>>> Internal Git Terms:

- Blob: contents of a file (without the file metadata)
- Tree: directory listing of blobs and other trees
- Commit: snapshot of the working tree (with metadata)
- Branch: named reference to a commit
- HEAD: generally a pointer to the current branch
- Working directory: filesystem directory with .git repo
- Repository: collection of commits (with other info)
- Staging area (or Index): playground for the next commit

>>> Authentication with GitHub Today for Linux

For Linux, you need to configure the local GIT client with a username and email address,

```
$ git config --global user.name "your_github_username"
```

```
$ git config --global user.email "your_github_email"
```

```
$ git config -l
```

Once GIT is configured, we can begin using it to access GitHub. Example:

```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

```
> Cloning into `YOUR-REPOSITORY`...
```

```
Username: <type your username>
```

```
Password: <type your password or personal access token (GitHub)>
```

Now cache the given record in your computer to remembers the token:

— Continued on next page —

>>> Authentication with GitHub Today for Linux (continued)

```
$ git config --global credential.helper cache
```

If needed, anytime you can delete the cache record by:

```
$ git config --global --unset credential.helper
```

```
$ git config --system --unset credential.helper
```

Now try to pull with -v to verify

```
$ git pull -v
```

Linux/Debian (Clone as follows):

```
git clone https://<tokenhere>@github.com/<user>/<repo>.git
```

>>> Authentication with GitHub Today for OSX

Click on the Spotlight icon (magnifying glass) on the right side of the menu bar. Type Keychain access and then press the Enter key to launch the app. In Keychain Access, search for github.com, find the internet password entry for github.com, and Edit or Delete the entry accordingly.

>>> Authentication with GitHub Today for Windows

Go to Credential Manager from Control Panel => Windows Credentials

Click on git:<https://github.com>

Click on Edit and replace the Password with your GitHub Personal Access Token

Click on Save

If you don't find git:<https://github.com>, click on Add a generic credential and enter:

- Internet address - git:https://github.com
- Your username and
- Your GitHub Personal Access Token as the password
- Click Ok