

vzw BiASC asbl

Belgian IT Academy Support Center
Improving the way people learn



Git Cookbook for DevOps, Network & Systems Management

DevOps & DevNet Associate

Yvan Rooseleer

Belgian IT Academy Support Center BiASC
Amsterdam 11 May and Geel 13 May 2022



Cisco Networking Academy

Topics

Cisco NetAcad DevNet Associate Course

3.3 – Version Control Systems

3.3.2 – Git

3.3.3 – Local vs. Remote Repositories

3.3.4 – What is Branching?

3.3.5 – GitHub and Other Providers

3.3.6 – Git Commands

3.3.7 – Adding and Removing Files

3.3.8 – Updating Repositories

3.3.9 – Branching Features

3.3.10 – .diff Files

3.3.11 – Lab – Software Version Control with Git

Git repositories are an integral part of the Cisco Networking Academy **DevNet Associate** Course.

Git is able to manage:

- Source code (software)
- Configuration scripts (systems, networks, security)
- Documentation

6.3 – Continuous Integration/Continuous Deployment (CI/CD)

6.3.4 – Example Build Job for Jenkins

6.3.6 – Lab – Build a CI/CD Pipeline Using Jenkins

6.3.7 – Project Activity 4: Automated Software Testing and Deployment

Managing Git (and CI/CD Pipelines)

Git Repositories

- Planning, creating and managing repositories
- Local and Remote repositories
- Updating repositories
- Adding and removing folders and files
- Comparing versions
- Logging
- Giving access to repositories
- Creating and managing branches and mergers
- Best practices
- Git Hands-on lab

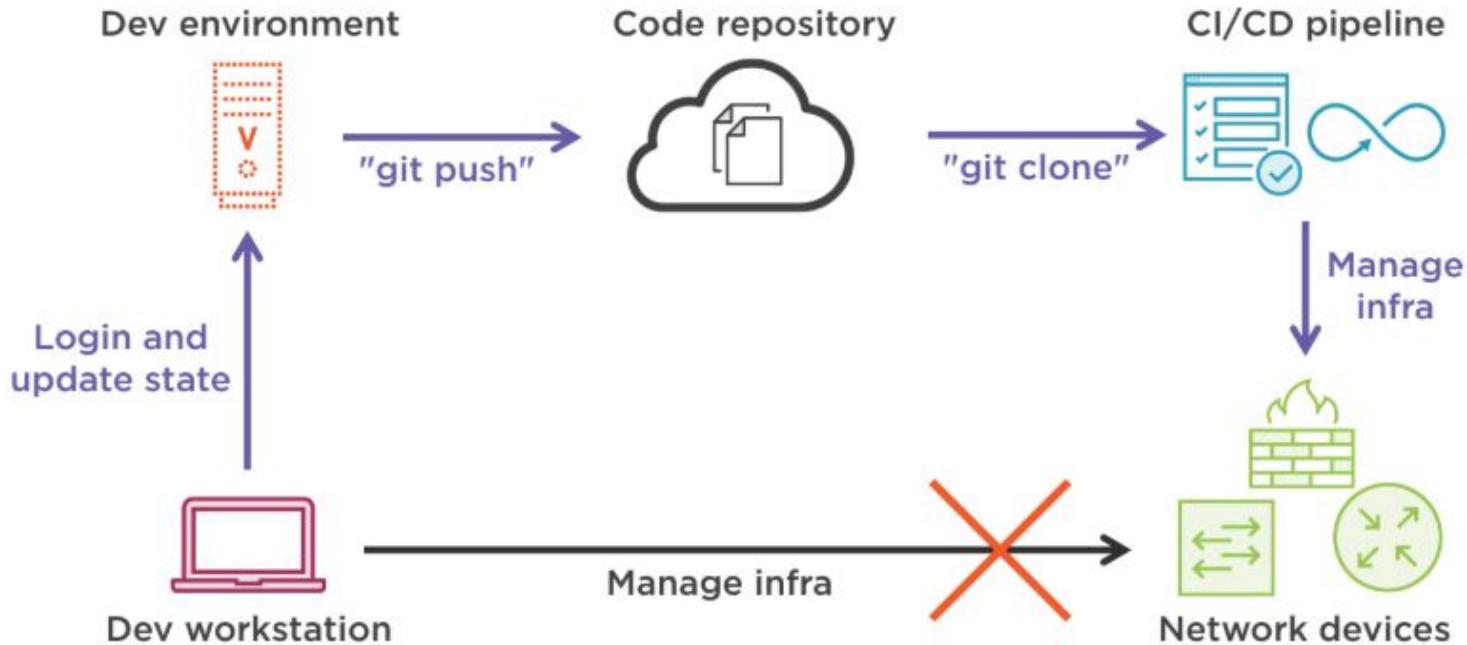
Continuous Integration/ Continuous Deployment (CI/CD) Pipelines => only FYI

- Pipeline scripting: Commit, Build, Test, Stage, Deploy
- Deployment of Python apps in an agile, containerized environment
- CI/CD Hands-on lab

Context

Network Programmability

Infrastructure as Code Operational Flow



Network Programmability -- NetDevOps Toolbag Overview

Continuous Integration					
Configuration Management					
Collaboration					
Working Environment					
Source/Image Control					
PaaS					
IaaS					

Five major phases of release and monitor

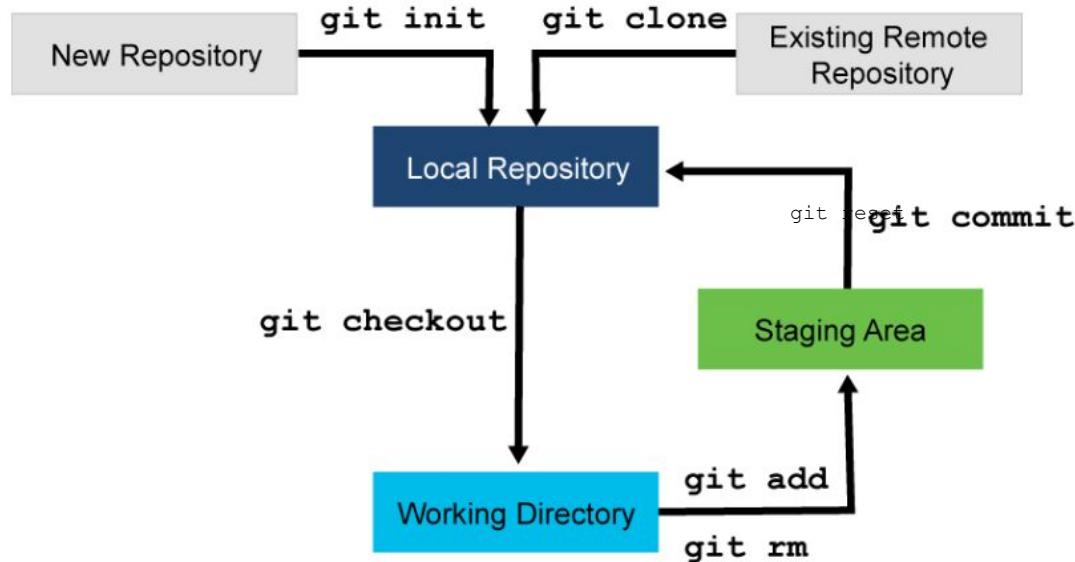


- Check-in source code such as .java files.
- Peer review new code
- Compile code
- Unit tests
- Style checkers
- Code metrics
- Create container images
- Integration tests with other systems
- Load testing
- UI tests
- Penetration testing
- Deployment to production environments
- Monitor code in production to quickly detect unusual activity or errors



Git Version Control

Version Control



Git

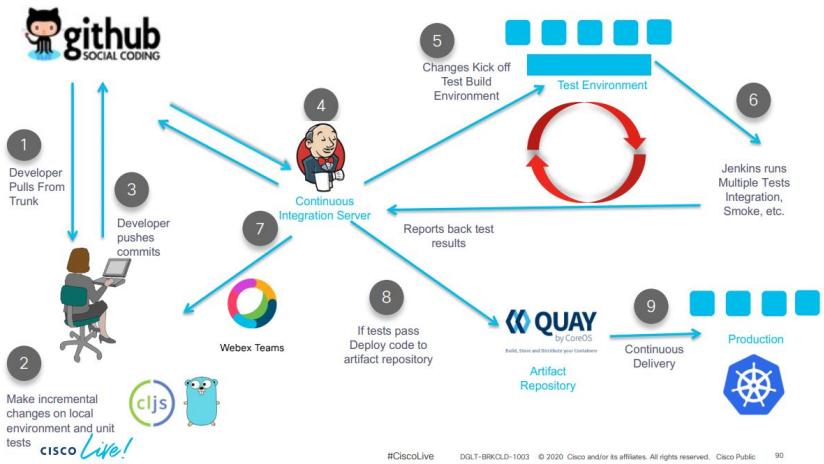
Open source

Distributed version control system

Stream of snapshots

Key difference between Git and other version control systems is that Git stores data as **snapshots** instead of differences (the **delta** between the current file and the previous version)

Repositories are often used in automation and CI/CD pipelines



Demo

Flask Microweb App - Chromium
 yroose/PythonExperiment | School Library | School Library API | Flask Microweb App
 localhost:5050

Called via IP Gateway: 127.0.0.1

Current date and time: 2022-04-29 19:39:13.191171

Preparing Git

Preparing Git => Cheat Sheet

Install

GitHub for Windows

<https://windows.github.com>

GitHub for Mac

<https://mac.github.com>

Git for All Platforms

<http://git-scm.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Configure tooling

Configure user information for all local repositories

\$ git config --global user.name "[name]"

Sets the name you want attached to your commit transactions

\$ git config --global user.email "[email address]"

Sets the email you want attached to your commit transactions

\$ git config --global color.ui auto

Enables helpful colorization of command line output

Create repositories

When starting out with a new repository, you only need to do it once; either locally, then push to GitHub, or by cloning an existing repository.

\$ git init

Turn an existing directory into a git repository

\$ git clone [url]

Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

The .gitignore file

Sometimes it may be a good idea to exclude files from being tracked with Git. This is typically done in a special file named `.gitignore`. You can find helpful templates for `.gitignore` files at github.com/github/gitignore.

Git Cooking

Create a new git repository on the command line

Example

```
echo "# PythonExperiments" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/yroosel/PythonExperiments.git
git push -u origin main
```

- `git branch -M` is a flag (shortcut) for --move --force
allows renaming the branch even if the new branch name already exists

Push a new git repository on the command line

Example

```
git remote add origin https://github.com/yroosel/PythonExperiments.git
git branch -M main
git push -u origin main
```

git push -u

is important for syncing your local git repository with your remote git repository when pushing to remote for the **first time**. The advantage is, you may use **git pull** without any arguments.

Useful Git Commands

Useful Git Commands

Setup	Tell git who you are <i>one-time setup</i>	git config --global user.name "your name" git config --global user.email your@email.com
Clone	Clone ("download") a git repository	git clone url
Status	Check the Status of your local repository	git status
Checkout A Branch	Create and Checkout a local Branch 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	Checks-out a file from the last commit.	git checkout filename

Git clone

When using `git clone` (from GitHub, or any source repository) the default name for the source of the clone is "origin".

Using `git remote show` will display the information about this remote name.

Example 1: `$ git remote show`

Response: origin

Here is a method to find the url of the remote repo

Example 2: `$ git remote get-url origin`

Response: https://github.com/yro/python_scripts.git

Remote Git Repository URL

Here is a method to find the url of the remote url for the name origin

Example \$ git config --get remote.origin.url

Response:

https://github.com/yroosel/python_scripts.git

Remote Git Repository URL (2)

Here is a method to find the url of the remote url both for fetching or for pushing

Example \$ git remote -v

Response:

origin https://github.com/yro/python_scripts.git (fetch)

origin https://github.com/yro/python_scripts.git (push)

Remote Git Remote Repository Details

Example `$ git remote show origin`

Response:

```
* remote origin
  Fetch URL: https://github.com/yro/python_scripts.git
  Push  URL: https://github.com/yro/python_scripts.git
  HEAD branch: main

  Remote branches:

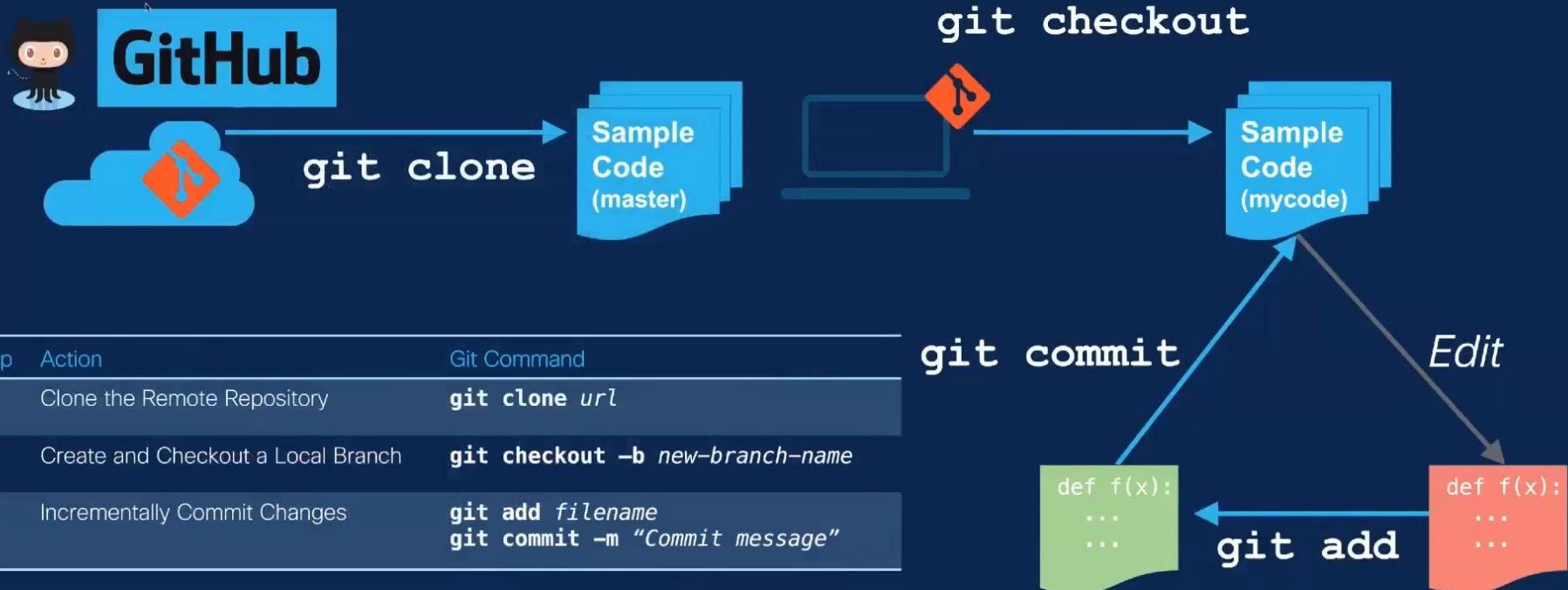
    main      tracked
    master    tracked

  Local branch configured for 'git pull':
    master merges with remote master

  Local ref configured for 'git push':
    master pushes to master (up to date)
```

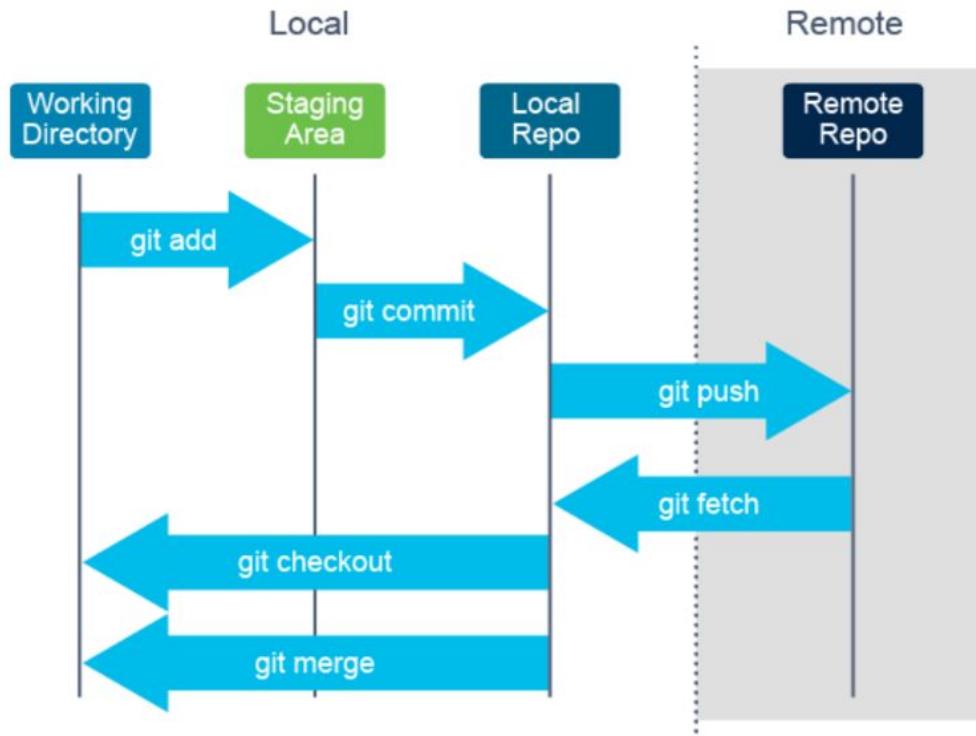
Git Workflow

DevNet Sample-Code Workflow

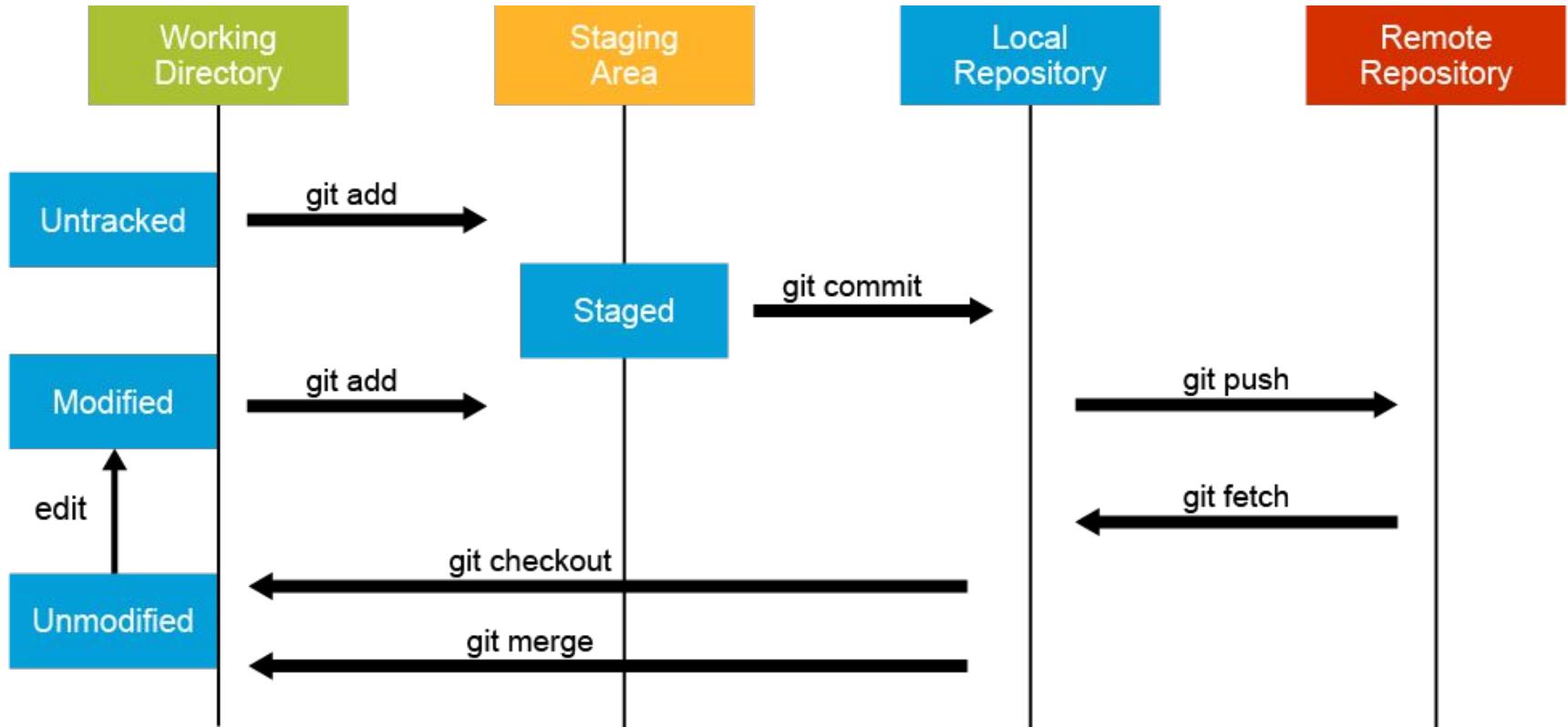


DEVASC Topics #1: Software Development and Design (15%)

- 1.8 Utilize common version control operations with Git

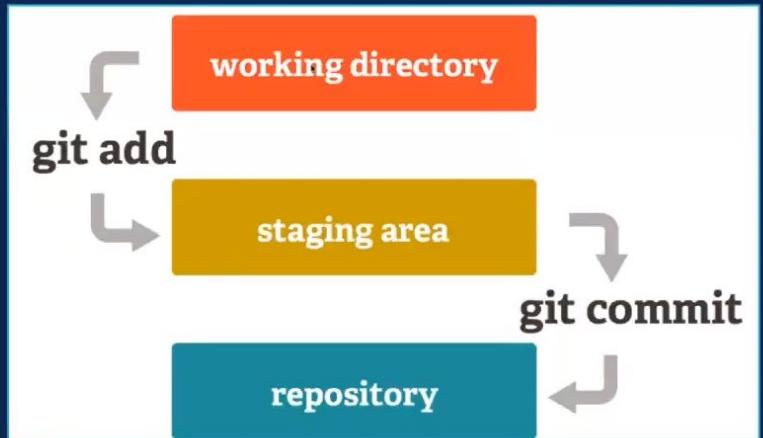


- git config --global user.name "Jozef J"**
git config --global user.email "jj@example.com"
- git init**
 - Setup a local git repository
- git add <files...>**
 - Add files to be tracked in the repository
- git rm <files...>**
 - Remove files from the repository tracking
- git commit -m "description of the commit"**
 - Commit the changes
- git push**
 - Send local repository to remote server
- git pull**
 - Pull data from the remote server
- git diff**
 - Display the differences since last commit



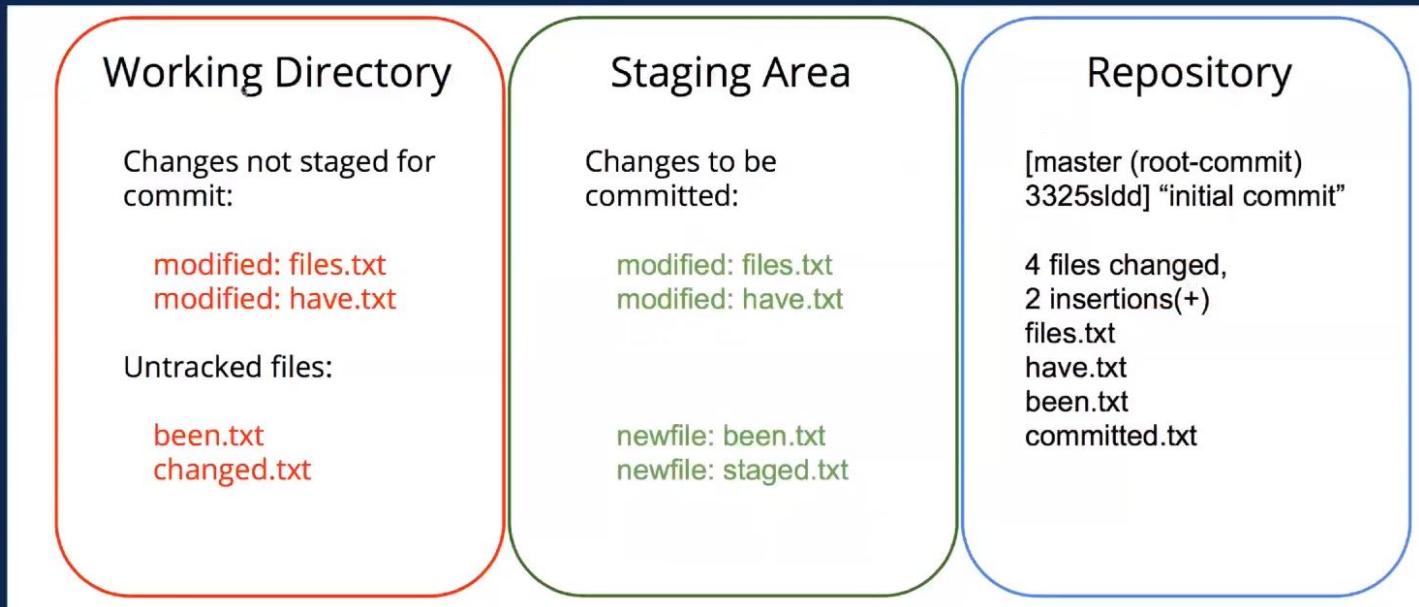
GIT ADD

- Add any files in your repository to git “stage”



```
$ git add .
```

STAGING AREA



GIT COMMIT

- Store your changes into a commit
- Saves all of your changes together / save point
- Commit does NOT push

```
$ git commit -m 'Initial commit'
```

Git Log: History of commits

```
commit 37895108669192892df1056cae020131c775facc (HEAD -> master, github/master)
Author: Ashley Roach <asroach@cisco.com>
Date: Thu Dec 7 16:47:54 2017 -0700
```

```
remove proxy
```

```
commit 4c51b91c3fe974f9b929c64ad7cb2be7a7ae85e9 (origin/master, origin/HEAD)
Merge: fe54f3b 34987c1
Author: Ashley Roach <asroach@cisco.com>
Date: Tue Feb 7 08:23:31 2017 -0700
```

```
Merge branch 'master' of wwwin-github.cisco.com:DevNet/sandbox-devbox
```

```
commit fe54f3b3ef8e96ad09924bb7c46ae7eb29d55034
Author: Ashley Roach <asroach@cisco.com>
Date: Tue Feb 7 08:19:49 2017 -0700
```

```
Add License
```

```
commit 34987c15e4e2deefda9b9d5ae2730ef672a49358
Author: asroach <asroach@cisco.com>
Date: Thu Jan 12 13:44:46 2017 -0700
```

```
Add prereqs
```

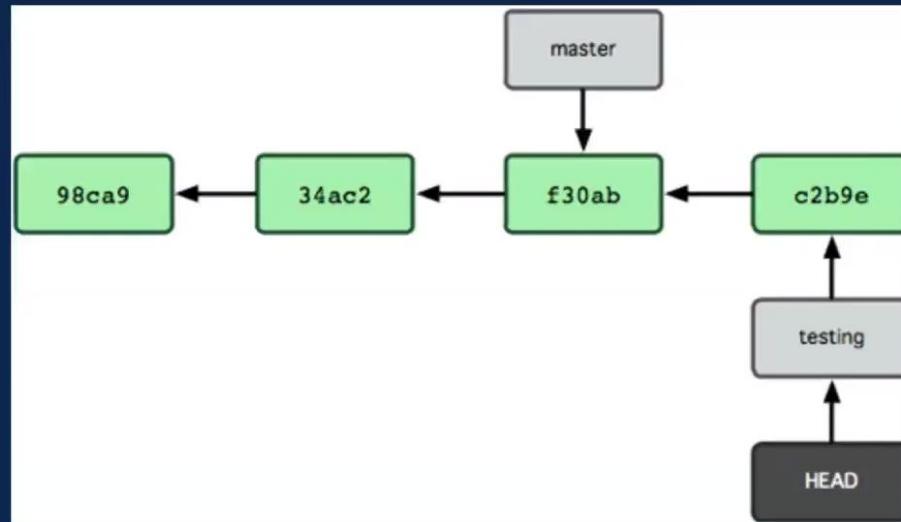
```
$ git log
```



BRANCHING: Your safe place

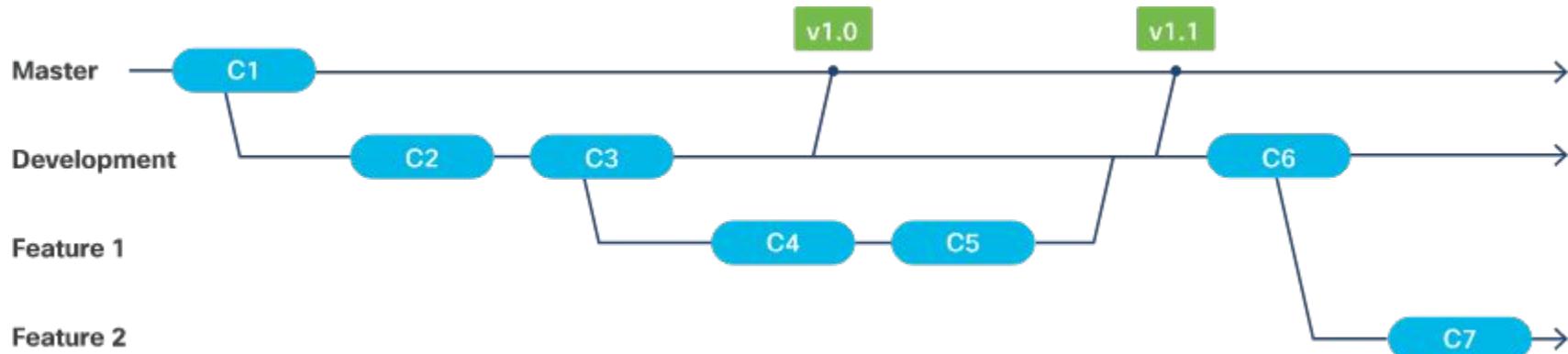
- Makes a pointer to your code
- Moves HEAD around

```
$ git branch <name>
$ git branch testing
$ git commit -m "new"
$ git checkout master
```



*Main is often used as a replacement of master

Git branching



- **Branching** is one of Git's major selling points
- Branching enables users to work on code **independently** without affecting the main code in the repository
- When a **repository** is created, the code is automatically put on a branch named **master**
- Users can have multiple branches and those are independent of each other as well

MERGING

- git merge <topic>
- You must be on the branch you want to merge INTO when you execute this command (e.g. master)

```
$ git merge <branch>
```

```
$ git merge testing
```

GOING BACKWARDS

- Generate a new commit that undoes all of the changes introduced in <commit>, then apply it to the current branch

```
$ git revert <commit>
```

SHARE YOUR CHANGES

- git push <destination> <branch>
- git push origin master

```
$ git remote add <name> <url>
$ git push <name> <branch>
$ git push origin master
```

*Main is often used as a replacement of master

* *Branches will “remember” the remote after first push.*

What's the
difference?

Comparing changes with git diff

- *Diffing* is a function that takes two input data sets and outputs the changes between them
- `git diff` is a multi-use Git command that when executed runs a diff function on Git data sources.
- These data sources can be commits, branches, files and more

Comparing changes with git diff (2)

Comparison input

```
$ diff --git a/diff_test.txt b/diff_test.txt
```

Metadata displayed by git

Internally git works with version hash identifiers and therefore shows (part of the) the hashes of the two files.

Example:

```
index 5c3b621..a0d8bf9 100644
```

Comparing changes with git diff (3)

Markers for changes

```
--- a/diff_test.txt
+++ b/diff_test.txt
```

These lines are a **legend** that assigns symbols to each diff input source.

In this case, changes from a/diff_test.txt are marked with a **---** and the changes from b/diff_test.txt are marked with the **+++** symbol.

Comparing changes with git diff (4)

Diff chunks

The main diff output is a **list of diff 'chunks'**

A diff only displays the sections of the file that have changes.

```
@@ -1 +1 @@
-this is a git diff test example
+this is a diff example
```

added.

This indicates a replacement or an update

Comparing changes with git diff (5)

Comparing files from two branches

```
$ git diff main new_branch ./diff_test.txt
```

To compare a specific file across branches, pass in the path of the file as the third argument to git diff

Practising git diff

CHANGED FILE

```
def add(n1,n2):  
    return n1+n2  
  
def subtract(n1, n2):  
    return n1-n2  
  
def multiply(n1, n2):  
    return n1*n2  
  
def divide(n1, n2):      ↪ line 8  
    return n1/n2  
  
def main():  
    print (add(100,10))  
  
if __name__ == "__main__":  
    main()
```

```
def add(n1,n2):  
    return n1+n2  
  
def subtract(n1, n2):  
    return n1-n2  
  
def multiply(n1, n2):  
    return n1*n2  
  
def divide(n1, n2):      ↪ line 8  
    if n2 != 0:  
        return n1/n2  
    else:  
        return ("Error: Division by Zero")  
  
def main():  
    print (add(100,10))  
    print (subtract(100,10))  
    print (multiply(100,10))  
    print (divide(100,10))
```

Practising git diff --B

```
>git diff
```

```
diff --git a/calc.py b/calc.py
index 2519d86..57a02fd 100644
--- a/calc.py
+++ b/calc.py
@@ -8,10 +8,16 @@ def multiply(n1, n2):
    return n1*n2

def divide(n1, n2):
-    return n1/n2
+    if n2 != 0:
+        return n1:n2
+    else:
+        return ("Error: Division by Zero")

def main():
    print (add(100,10))
+    print (subtract(100,10))
+    print (multiply(100,10))
+    print (divide(100,10))
```

Python
Code Samples
on GitHub

CiscoDevNet

python_code_samples_network

A collection of Python Code Samples for Network Management. Includes samples to run on-box and off-box.

⌚ 126 ★ 370 💾 273

[View on GitHub](#)**OWNER**

CiscoDevNet

CONTRIBUTOR xorrkaz hpreston krishna426426**CATEGORIES**

Network Automation with Python Code Samples

Downloading git existing repository

```
$ git config --global user.email "yro@biasc.com"
```

```
$ git config --list
```

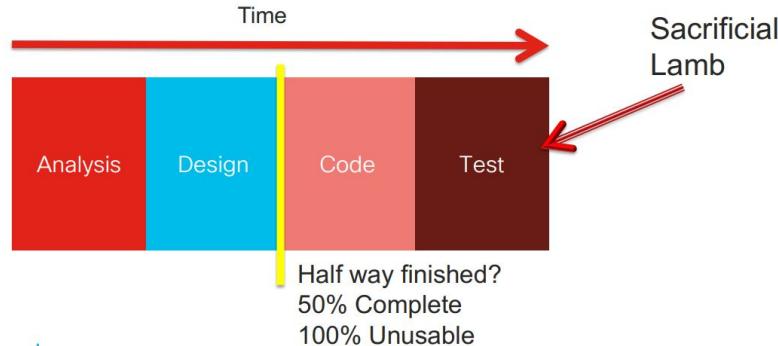
```
$ git init
```

```
$ git clone https://github.com/CiscoDevNet/python\_code\_samples\_network
```

```
$ git status
```

Pipelines

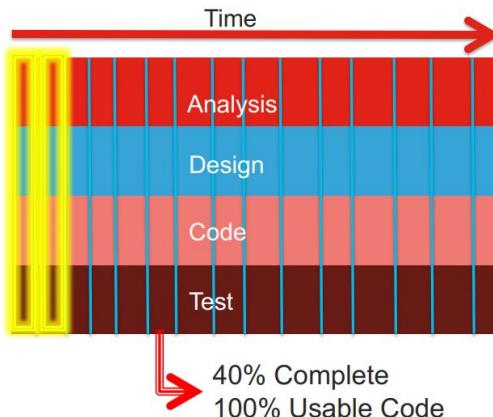
Before Agile Methodologies



CISCO Live!

#CiscoLive DGLT-BRKCLD-1003 © 2020 Cisco and/or its affiliates. All rights reserved. Cisco Public 20

Benefits of Agile Development



CISCO Live!

#CiscoLive DGLT-BRKCLD-1003 © 2020 Cisco and/or its affiliates. All rights reserved. Cisco Public 21

CISCO Live!

Agile & Scrum

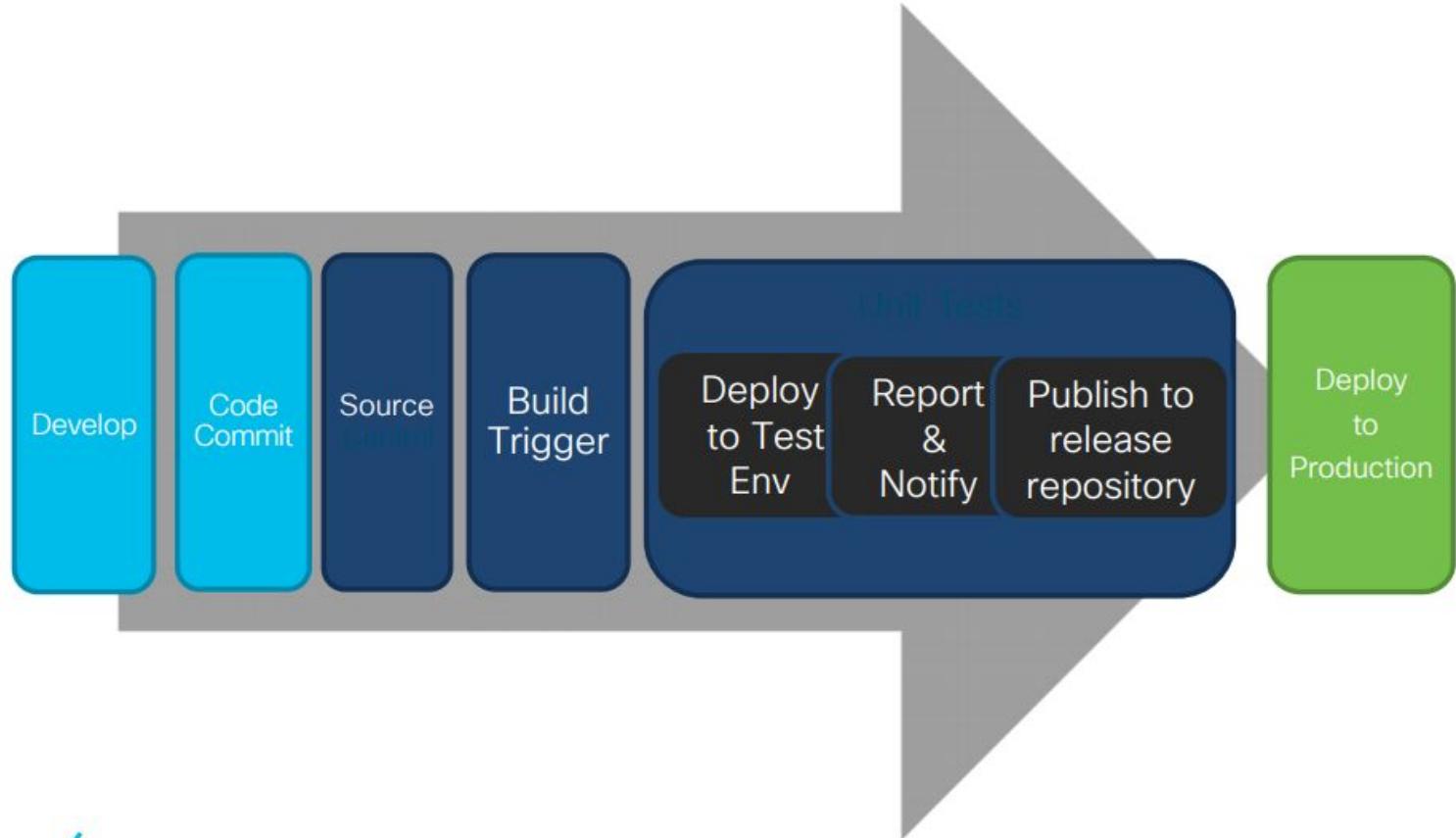
=> More Releases

SCRUM Agile Project Management



#CiscoLive DGLT-BRKCLD-1003 © 2020 Cisco and/or its affiliates. All rights reserved. Cisco Public 30

Continuous Deployment



Tips for Deploying Python Applications

Create a Python Virtual Environment

Install the necessary libraries and functionalities

Create a git repository

Push, Pull or Clone when necessary

Create branches and test code before merging into main branch

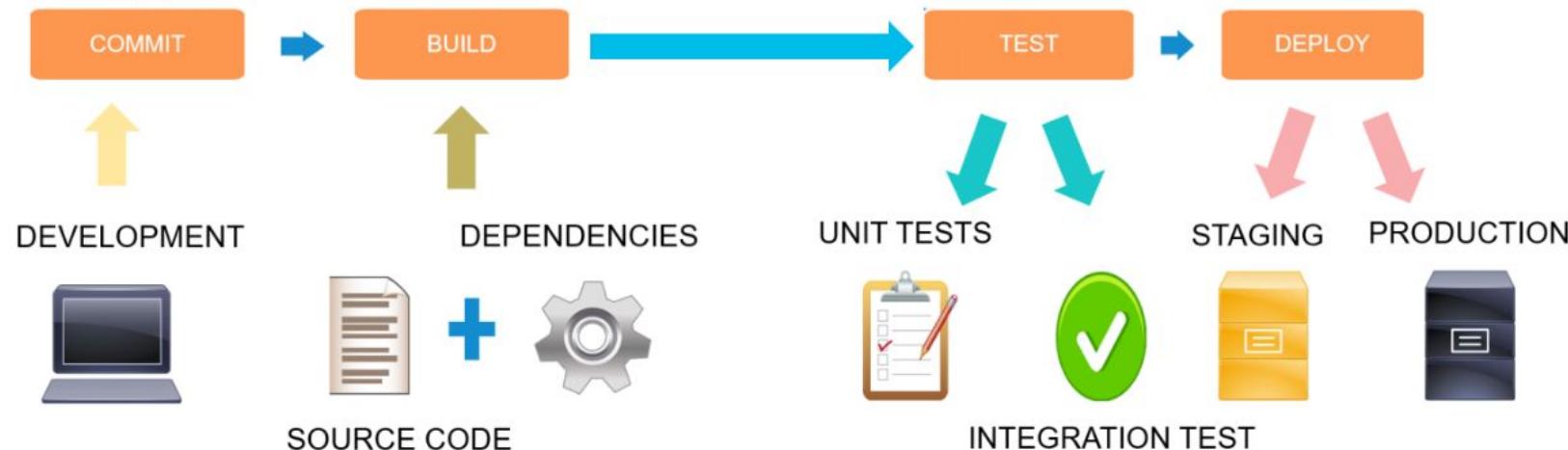
Use pipelines to automate deployment => CI/CD

DEVASC Topics #4: Application Deployment and Security (15%)

- 4.4 Describe components for a CI/CD pipeline in application deployments

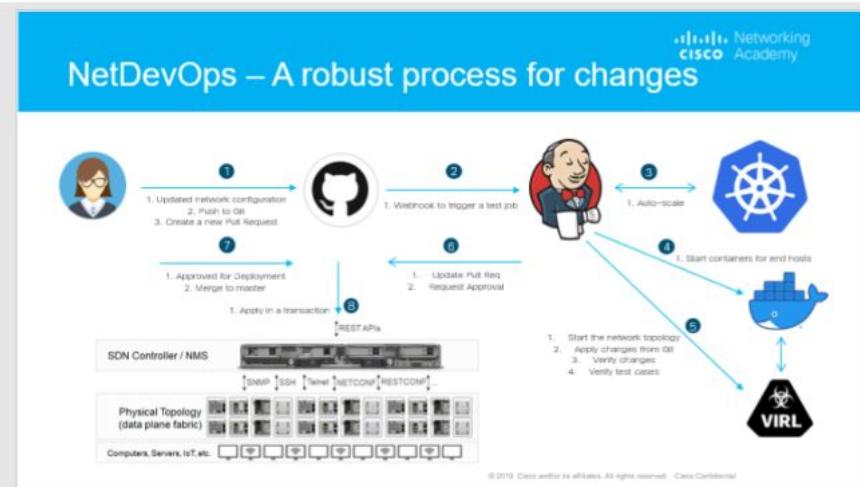
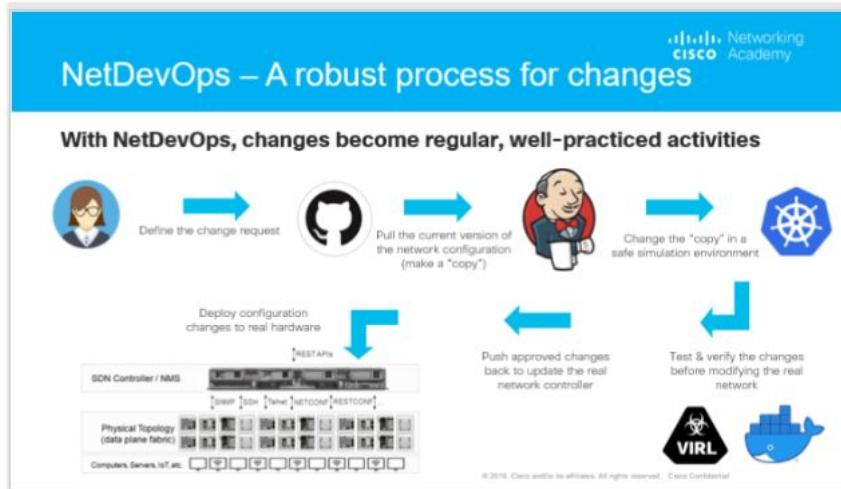
Continuous Integration, Continuous Deployment

A common pipeline process:



DEVASC Topics #5: Infrastructure and Automation (20%)

- 5.4 Describe the components and benefits of CI/CD pipeline in infrastructure automation

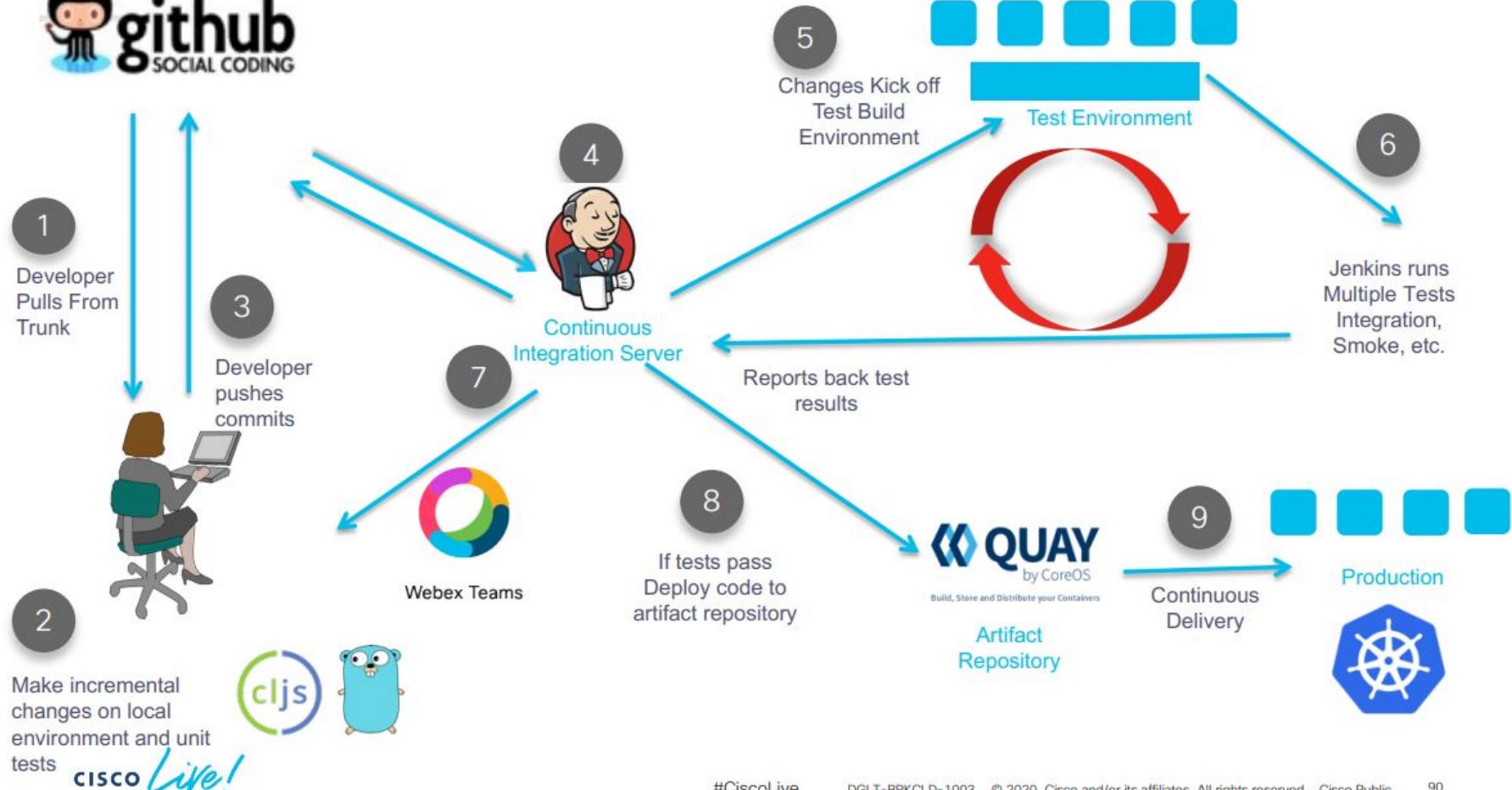




General best practices used by Amazon developers

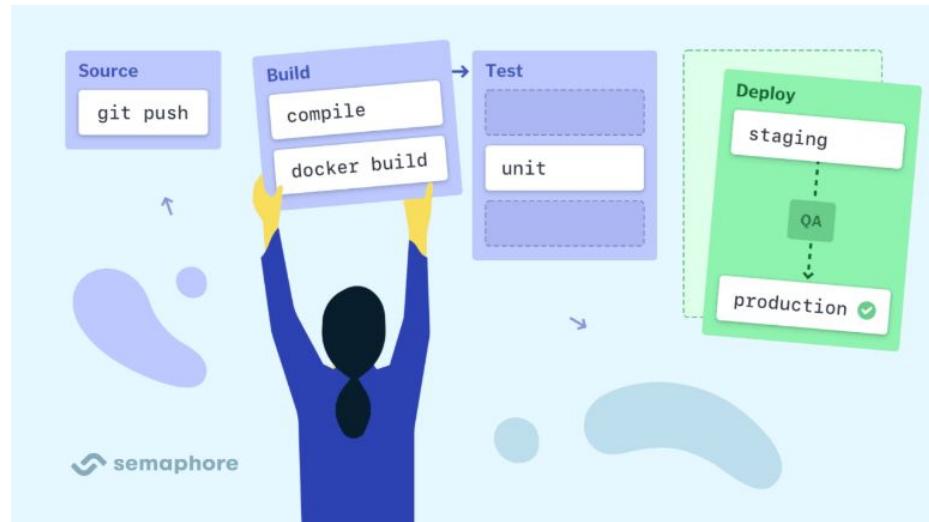
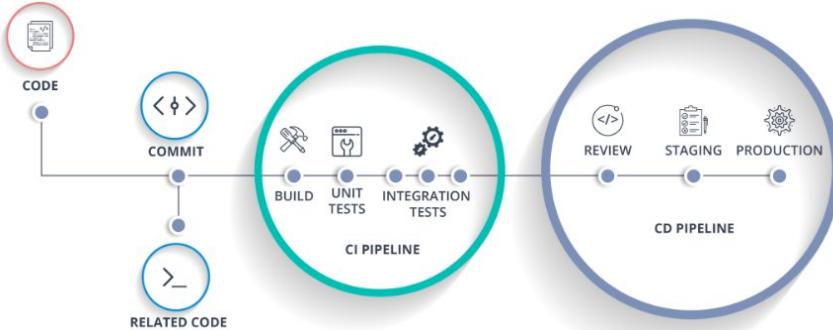
- CI/CD is a MUST!
 - Commit frequently
 - Builds on every commit
 - Build once in a given execution flow
 - Deploy to a running environment for further testing
- Everything that is code (application, infrastructure, documentation) goes into a repository
 - If its not in a repository, it doesn't go into production environments!
- Start with continuous delivery (“gated” promotion) and build up to continuous deployment once evidence of a high-level of excellence in testing is clear
- Deploy to canaries, test, deploy to an AZ, test, deploy to a Region, test



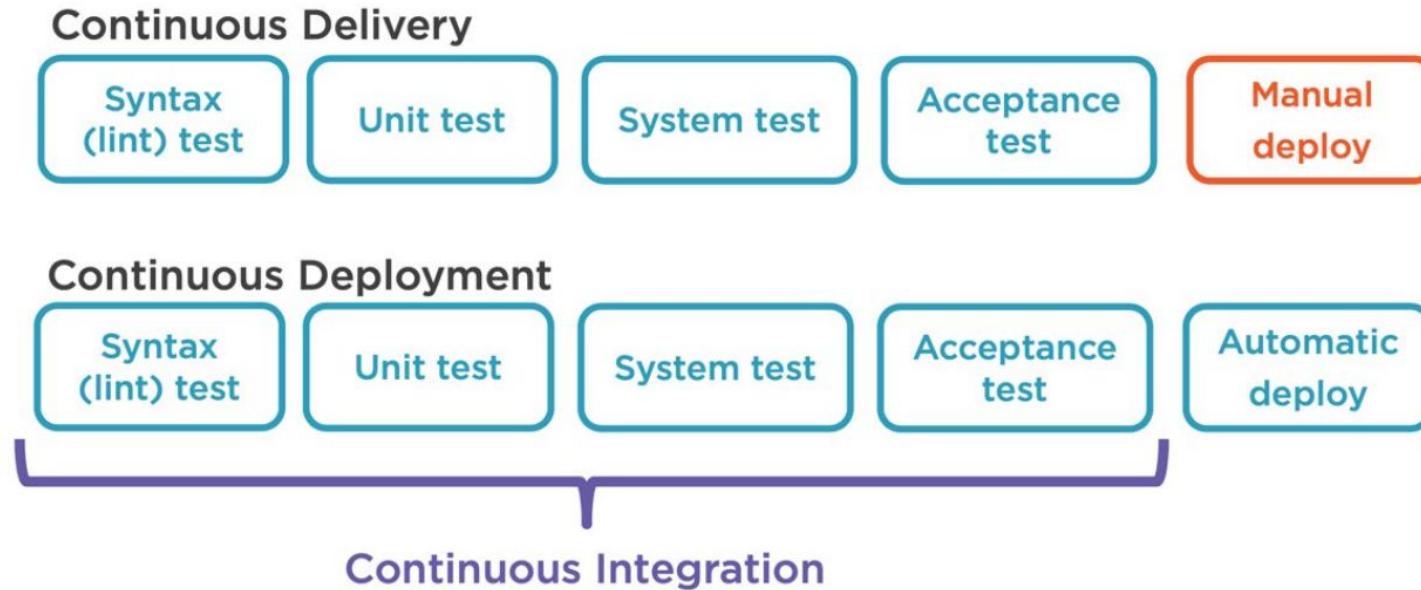


DevASC 4.0 -- Continuous Integration/ Continuous Delivery

CI/CD Pipeline



CI/CD Pipelines – DevOps, NetDevOps, DevSecOps



CI/CD Platforms: The Automated Software Bakery

Classic Mode

- Centralized Configuration
- Responsibility of CI/CD administrator
- Examples:
 - Jenkins, Microsoft TFS



Emerging Mode

- Configuration pushed into the Repositories not centralized
- Responsibility of Developers
- Examples:
 - [GitLab](#)
 - Drone.io, Travis CI, Circle CI

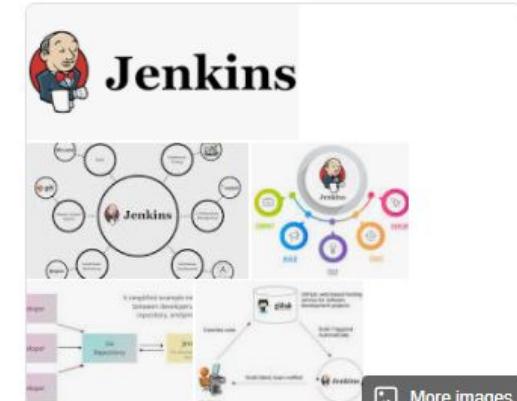


Jenkins Pipeline => Stages

The main role of Jenkins is to execute the stages of a **pipeline** and to verify/ test the results

Jenkins Pipeline is a suite of **plugins** that supports implementing and integrating continuous delivery pipelines into Jenkins. Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code"

There are plugins for: **Bash, Git, Docker, Ansible, and many more ...**



Jenkins

Software

Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. [Wikipedia](#)

Repository: github.com/jenkinsci/jenkins

License: MIT License

Platform: Java 8, Java 11

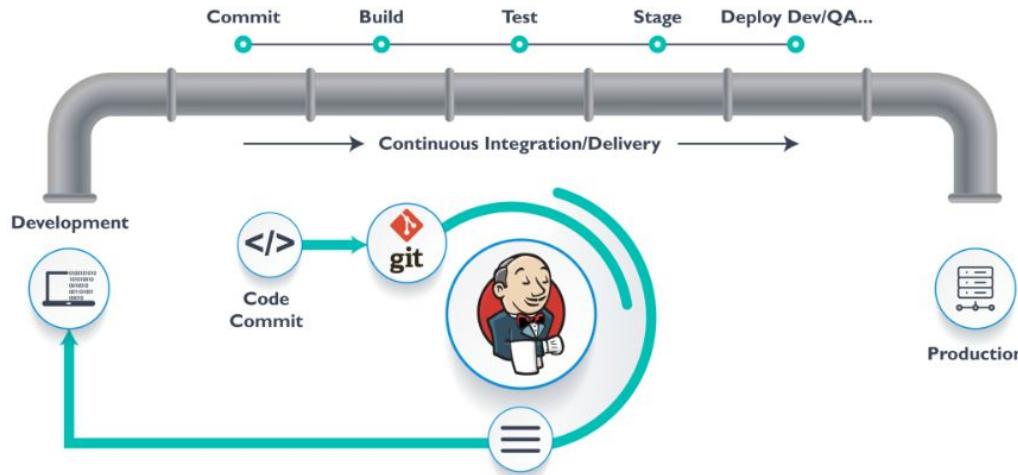
Initial release date: February 2, 2011

Programming language: Java

Developer: CloudBees, Kohsuke Kawaguchi

Original author: Kohsuke Kawaguchi

DevASC 4.0 -- Components of a CI/CD pipeline



A screenshot of the Jenkins UI titled "Build Pipeline". The pipeline consists of three stages: Job1, Job2, and Job3. Job1 is yellow and has a "Run" button highlighted with a red border. Job2 is light blue and shows "N/A" for its status. Job3 is also light blue and shows "N/A" for its status. The Jenkins header includes links for "Jenkins", "Ci CD Pipeline", "edureka", and "log out".

Jenkins Jobs

Preparation

Retrieve a version of the code from GitHub

Build

Run the build script

Testing

Test the build to ensure that it's working properly

Pipeline in three stages

Preparation

Build

Testing

Stages are run sequentially. If an early stage fails, the pipeline stops

Create a Jenkins job or pipeline: new item

Jenkins

Dashboard > All >

Enter an item name

| BuildFlaskAppJob

» Required field

 **Freestyle project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

 **Pipeline**

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

search

?

3

admin



General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions

Source Code Management

 None Git

Repositories

Repository URL

Credentials



Use Personal Access Token
In place of password

[Advanced...](#)[Add Repository](#)

Branches to build

Branch Specifier (blank for 'any')

[Add Branch](#)

Repository browser

(Auto)

Additional Behaviours

[Save](#)[Apply](#)

General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions

Build**Execute shell**Command **bash ./sample-app.sh**

See the list of available environment variables

Add build step ▾**Post-build Actions****Add post-build action ▾****Save****Apply**

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

echo "FROM python" > tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile
echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY sample_app.py /home/myapp/" >> tempdir/Dockerfile
echo "EXPOSE 5059" >> tempdir/Dockerfile
echo "CMD python /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir
docker build -t sampleapp .
docker run -t -d -p 5099:5059 --name samplerunning sampleapp
docker ps -a
```



General Build Triggers Advanced Project Options

Pipeline

Advanced...

Pipeline

Definition

Pipeline script

```
1  node {  
2    stage('Preparation') {  
3      catchError(buildResult: 'SUCCESS') {  
4        sh 'docker container stop samplerunning'  
5        sh 'docker container rm samplerunning'  
6      }  
7    }  
8    stage('Build') {  
9      build 'BuildFlaskAppJob'  
10   }  
11   stage('Results') {  
12     build 'TestFlaskAppJob'  
13   }  
14 }  
15  
16 }
```

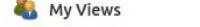
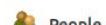
stages Use Groovy Sandbox[Pipeline Syntax](#)**Save****Apply**

Jenkins

Jenkins



New Item



Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

All

+

S	W	Name	Last Success	Last Failure	Last Duration	
		BuildFlaskAppJob	6 min 2 sec - #6	N/A	2.9 sec	
		SamplePipeline	6 min 28 sec - #1	N/A	35 sec	
		TestFlaskAppJob	5 min 52 sec - #4	21 min - #2	66 ms	

Icon: S M L

Legend

Atom feed for all

Atom feed for failures

Atom feed for just latest builds



Jenkins > SamplePipeline >

Back to Dashboard

Status

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

Build History

trend ^

find

#1 Nov 7, 2020 5:52 PM

Atom feed for all Atom feed for failures

Pipeline SamplePipeline

exploring pipelines

edit description

Disable Project



Recent Changes

Stage View



Permalinks

- Last build (#1), 2 min 34 sec ago
- Last stable build (#1), 2 min 34 sec ago
- Last successful build (#1), 2 min 34 sec ago
- Last completed build (#1), 2 min 34 sec ago

SamplePipeline #1 Console [Jenkins] - Mozilla Firefox

Sample Pipeline #1 Cons X Sample app x +

localhost:8080/job/SamplePipeline/lastBuild/console

Jenkins

search admin log out

Back to Project

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#1'

Replay

Pipeline Steps

Workspaces

Console Output

```
Started by user admin
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/SamplePipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Preparation)
[Pipeline] catchError
[Pipeline] {
[Pipeline] sh
+ docker container stop samplerunning
samplerunning
[Pipeline] sh
+ docker container rm samplerunning
samplerunning
[Pipeline] }
[Pipeline] // catchError
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] build (Building BuildFlaskAppJob)
Scheduling project: BuildFlaskAppJob
Starting building: BuildFlaskAppJob #6
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Results)
[Pipeline] build (Building TestFlaskAppJob)
```

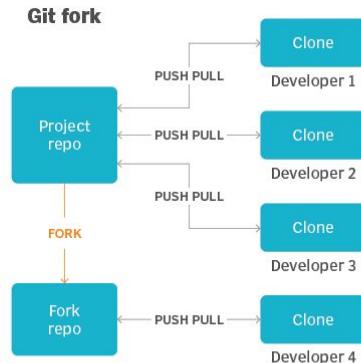
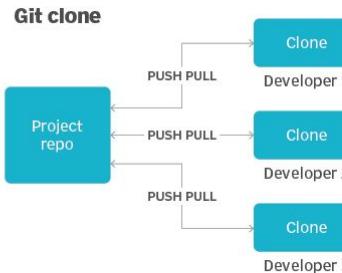
Git clone vs fork

Git fork vs. clone: What's the difference?

Any public Git repository can be forked or cloned. A fork creates a completely independent copy of Git repository. In contrast to a fork, a Git clone creates a linked copy that will continue to synchronize with the target repository.

Git clone vs. fork

Developers who work on a common codebase will clone the repository and then perform push and pull operations to synchronize their changes. In contrast, a fork creates a new codebase and updates to the fork are not synchronized with the original repo.



Review

What's the aim of the following git commands?

\$ git config --global user.email "yro@biasc.com"

\$ git config --list

\$ git checkout -b test origin /main

\$ git branch

\$ git fetch

\$ git diff

\$ git remote -v

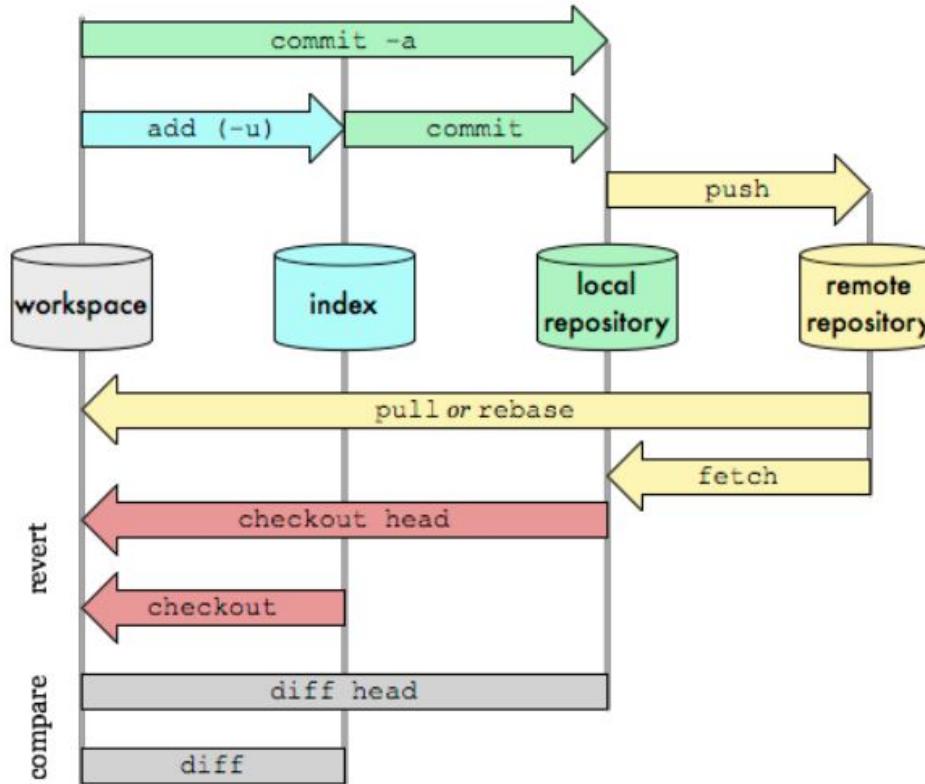
\$ git add .

\$ git status

Git Cheat Sheet

Git Data Transport Commands

<http://csteelle.com>



Need a Place to Store Code

- A place to store current and past versions of code
 - Ability to merge, branch, fork
 - Ability to see who has changed which file
 - Highly available service
 - Ability to revert to prior versions
 - Public or Private / OnPrem / OffPrem
 - Social



Infrastructure Automation: IaC Tooling Comparison

Infrastructure as Code, Repositories => **Git**

Infrastructure as Code, Cloud Provisioning => **Terraform**

Infrastructure as Code, Configuration management => **Ansible**

Continuous Integration/Deployment => **Docker, Kubernetes**

Continuous Integration/Pipelines => **Jenkins, Github Actions, Gitlab**

Config/Secret Management => **Consul, etcd, Vault**

Monitoring => **Prometheus**

Connecting, managing, and securing microservices (service mesh) => **Istio**

Devops Terms & Concepts

Repository => git

A collection of **scripts, programs and documents** is called a repository.

It is possible to **manage** and **store** files locally or on github. **One repo per application. One application per repository.**

Image => docker build

It is possible to create a **(micro)service** for a specific application.

An **image** is the file containing all the necessary artifacts for that **(micro)service**.

Typically, we use a **dockerfile** to create an image.

An image is **stored** on a file system or on docker hub. **One image per application. One application per image.**

Container => docker run, docker compose, kubernetes

A container is a **(micro)service** available to users by means of an **IP address and a Port**. In order to make an application available it is necessary to run or deploy the image. **As many services as necessary for one application.**

Pipeline => jenkins

A jenkins pipeline consists of stages in order to **deploy a (micro)service**. The pipeline starts with **fetching** the relevant version of the application code from a git repository. Then a docker image is created and stored locally (or on github). In the next phase it is possible to make the application image available (run) to make sure that users can connect to it.

References

Git Cheat Sheet - 2

Install

GitHub for Windows

<https://windows.github.com>

GitHub for Mac

<https://mac.github.com>

Git for All Platforms

<http://git-scm.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Configure tooling

Configure user information for all local repositories

\$ git config --global user.name "[name]"

Sets the name you want attached to your commit transactions

\$ git config --global user.email "[email address]"

Sets the email you want attached to your commit transactions

\$ git config --global color.ui auto

Enables helpful colorization of command line output

Create repositories

When starting out with a new repository, you only need to do it once; either locally, then push to GitHub, or by cloning an existing repository.

\$ git init

Turn an existing directory into a git repository

\$ git clone [url]

Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

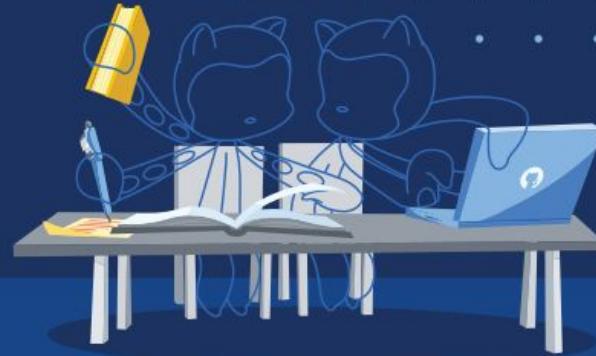
The .gitignore file

Sometimes it may be a good idea to exclude files from being tracked with Git. This is typically done in a special file named `.gitignore`. You can find helpful templates for `.gitignore` files at github.com/github/gitignore.

Git Cheat Sheet - 1

GitHub

Git Cheat Sheet



Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

Git Cheat Sheet - 3

Branches

Branches are an important part of working with Git. Any commits you make will be made on the branch you're currently "checked out" to. Use `git status` to see which branch that is.

\$ git branch [branch-name]

Creates a new branch

\$ git checkout [branch-name]

Switches to the specified branch and updates the working directory

\$ git merge [branch]

Combines the specified branch's history into the current branch. This is usually done in pull requests, but is an important Git operation.

\$ git branch -d [branch-name]

Deletes the specified branch

Synchronize changes

Synchronize your local repository with the remote repository on GitHub.com

\$ git fetch

Downloads all history from the remote tracking branches

\$ git merge

Combines remote tracking branch into current local branch

\$ git push

Uploads all local branch commits to GitHub

\$ git pull

Updates your current local working branch with all new commits from the corresponding remote branch on GitHub.

`git pull` is a combination of `git fetch` and `git merge`

Git Cheat Sheet - 4

Make changes

Browse and inspect the evolution of project files

\$ git log

Lists version history for the current branch

\$ git log --follow [file]

Lists version history for a file, including renames

\$ git diff [first-branch]...[second-branch]

Shows content differences between two branches

\$ git show [commit]

Outputs metadata and content changes of the specified commit

\$ git add [file]

Snapshots the file in preparation for versioning

\$ git commit -m "[descriptive message]"

Records file snapshots permanently in version history

Redo commits

Erase mistakes and craft replacement history

\$ git reset [commit]

Undoes all commits after [commit], preserving changes locally

\$ git reset --hard [commit]

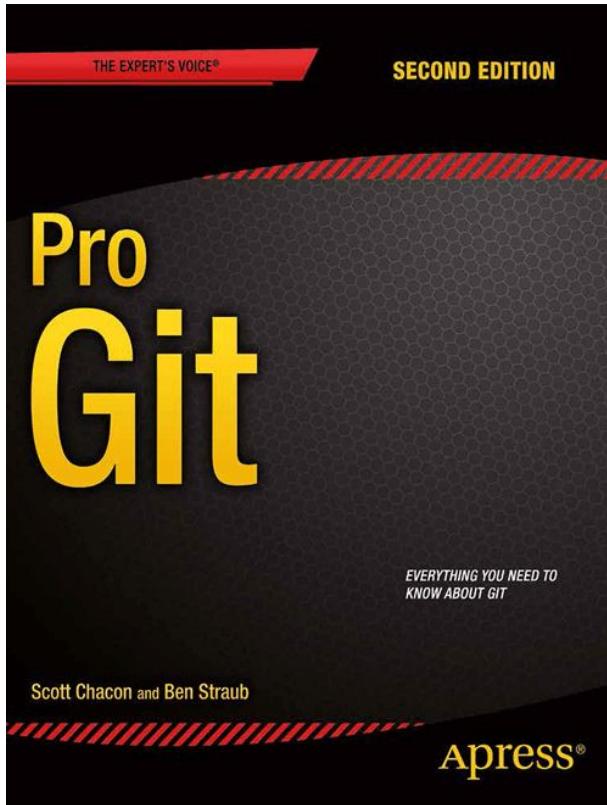
Discards all history and changes back to the specified commit

CAUTION! Changing history can have nasty side effects. If you need to change commits that exist on GitHub (the remote), proceed with caution. If you need help, reach out at github.community or contact support.

Git Cheat Sheet - Glossary

- **git**: an open source, distributed version-control system
- **GitHub**: a platform for hosting and collaborating on Git repositories
- **commit**: a Git object, a snapshot of your entire repository compressed into a SHA
- **branch**: a lightweight movable pointer to a commit
- **clone**: a local version of a repository, including all commits and branches
- **remote**: a common repository on GitHub that all team member use to exchange their changes
- **fork**: a copy of a repository on GitHub owned by a different user
- **pull request**: a place to compare and discuss the differences introduced on a branch with reviews, comments, integrated tests, and more
- **HEAD**: representing your current working directory, the HEAD pointer can be moved to different branches, tags, or commits when using git checkout

Pro Git book, written by Scott Chacon and Ben Straub



Getting Started
Git Basics
Git Branching
Git on the Server
Distributed Git
GitHub
Git Tools
...

References

References

Cisco Networking Academy, **DevNet Associate 1.0 (DEVASC)**

URL: www.netacad.com

Login necessary

C. Jackson, **A Practical Introduction to DevOps Practices and Tools**,

DGTL-BRKCLD-1003, CiscoLive 2020, URL: [Hyperlink](#)

Login necessary

H. Preston, **Overview of Git, Introduction to Version Control**, Lesson 2,

Cisco Learning Network and DevNetOctober 2020, URL: [Hyperlink](#),

Login necessary

CiscoDevNet, **python_code_samples_network**, Cisco DevNet, GitHub,

URL: [Hyperlink](#)

References 2

A. Khan, **DevOps Culture at Amazon**, Amazon Web Services, 47.01, 2 JUL 2018, YouTube, URL: <https://www.youtube.com/watch?v=mBU3AJ3j1rq>

Atlassian BitBucket, **Comparing changes with git diff**
URL: URL: <https://www.atlassian.com/git/tutorials/saving-changes/git-diff>

Semaphore, CI/CD Pipeline: A Gentle Introduction, URL: [Hyperlink](#)

S. Chacon & B. Straub, **Pro Git Book**, 2nd Edition (2014),
URL: <http://com/book/en/v2/git-scm>

Done