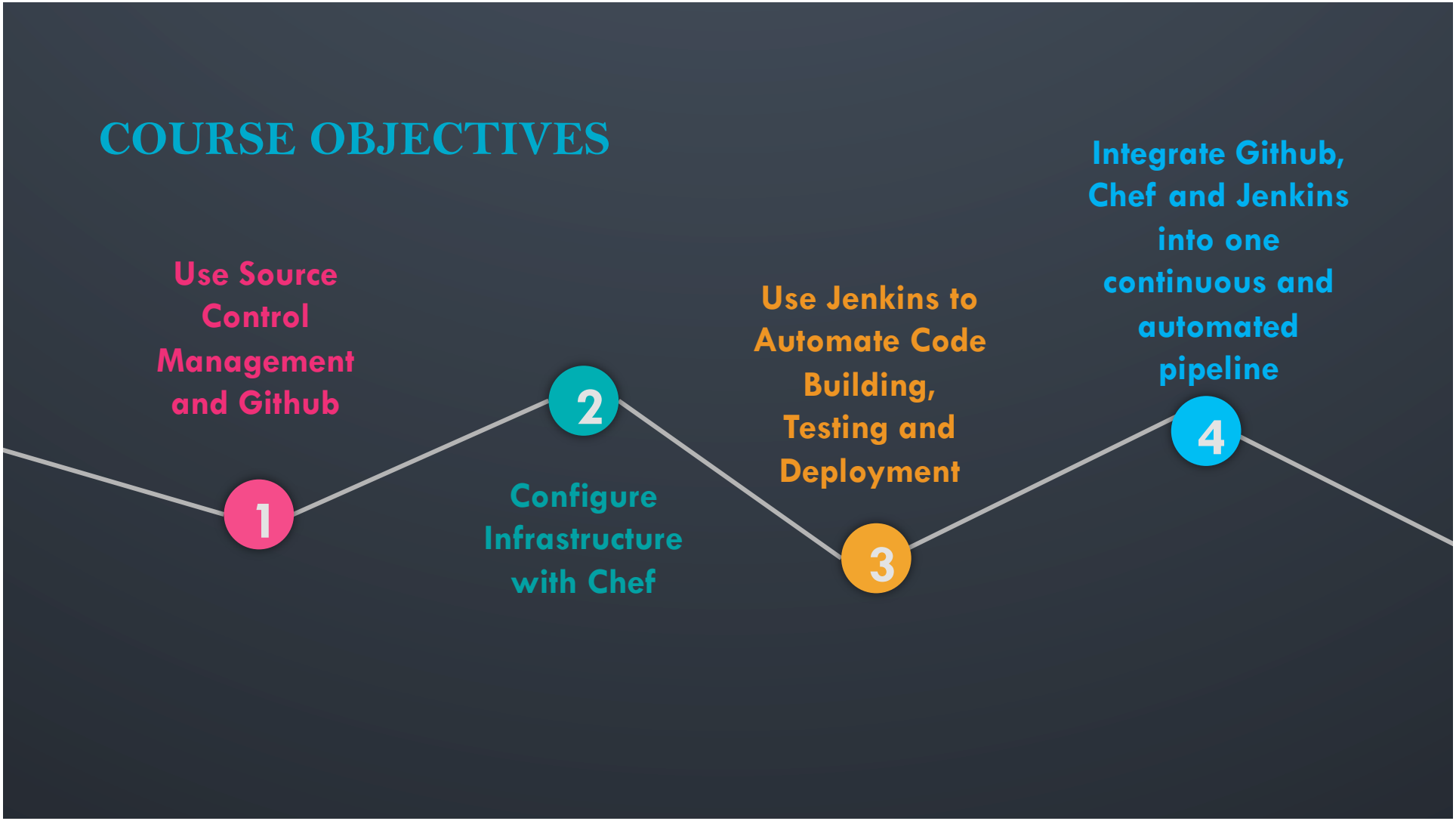
DevOps Pipeline

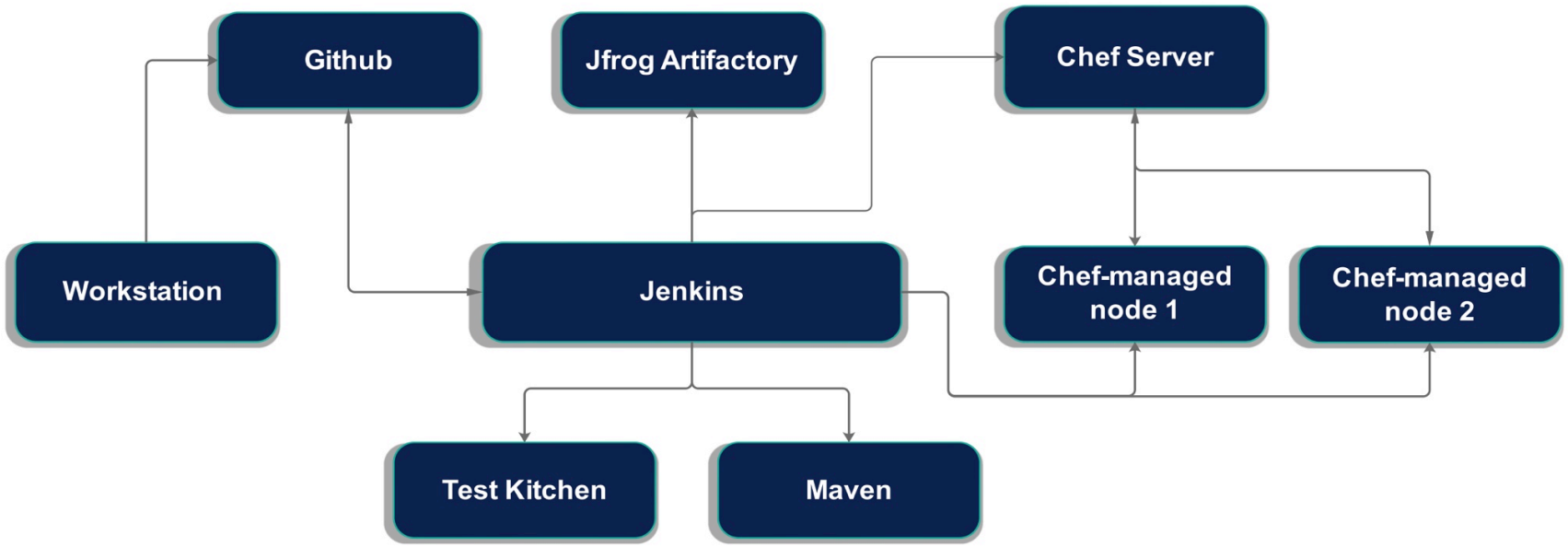
Git - Chef – Jenkins

CLASS OVERVIEW

- Course Objectives
- Class logistics
- Detailed Agenda
- Introductions
- Lab environment

COURSE OBJECTIVES




Github
Jfrog Artifactory
Chef Server
Workstation
Jenkins
Chef-managed node 1
Chef-managed node 2
Test Kitchen
Maven

CLASS LOGISTICS

- Start time
- Breaks
- Lunch
- End time
- Facilities

DETAILED AGENDA

GIT & GITHUB

- Git purpose and Workflow
- Installation & configuration
- Getting help with git
- Basic git commands
  - Remote, status, add, commit, push, log, diff
- Creating and checking out branches
- Creating a Github repo (repository)
- Accessing private repos with SSH keys
- Pull requests
- Merging and deleting branches

CHEF CONFIGURATION MANAGEMENT

- Purpose and Use cases
- Architecture and call flow
- Resources, recipes & cookbooks
- Policyfiles
- Integration testing - Inspec and Test Kitchen
- Chef Server as centralized management
- Creating a local chef-repo
- Bootstrapping Chef-managed nodes

CHEF

JENKINS

- Purpose & history
- Jenkins architecture
- Initializing a Jenkins Master
- Plugins
- Projects / jobs
- Freestyle UI jobs
- Pipeline jobs - CI/CD as Code
  - CI/CD = Continuous Integration / Continuous Deployment
- Declarative versus Scripted pipelines
- Views and folders
- Managing credentials and secrets

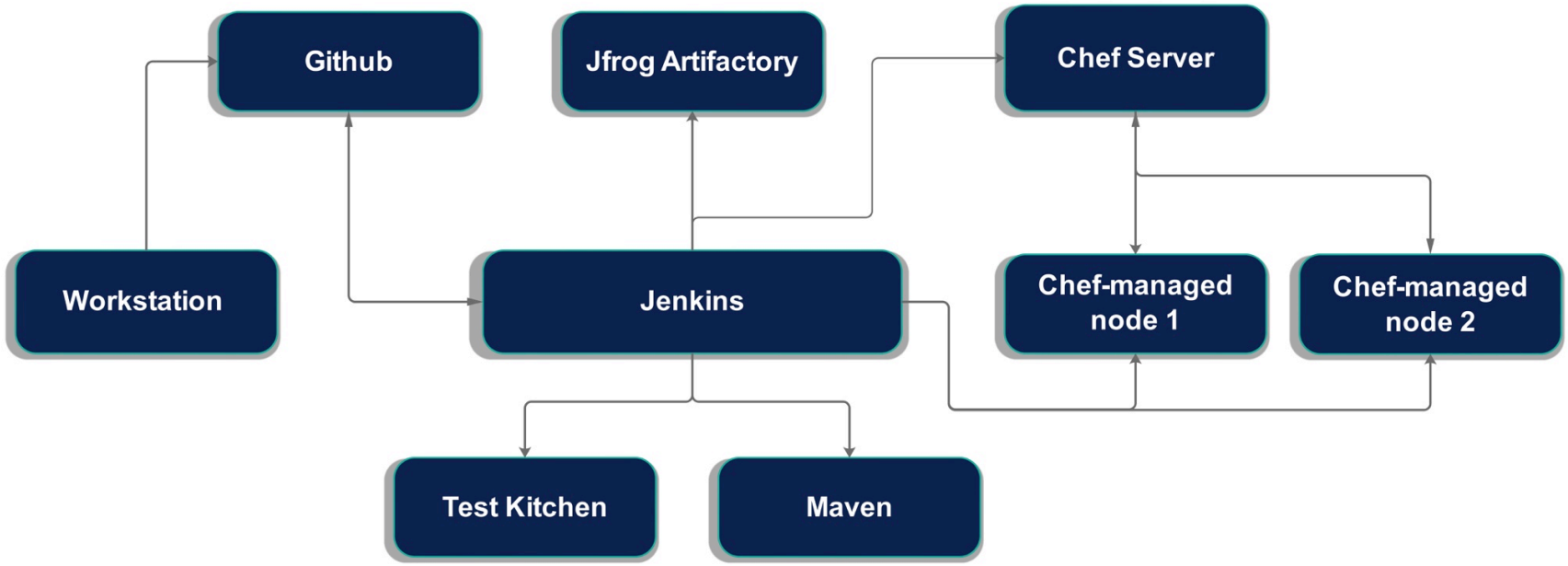
JENKINS

- Distributing workloads - Master and Agent nodes
- Integrating with Source Control Management: Github
- Triggers: Scheduled Polling and Github Webhooks
- Notifications: Slack and SMTP Email
- Requiring human input and approval
- Automated cookbook testing with Test Kitchen
- Jenkins Integration with Chef Server
- Continuous Deployment of Chef cookbooks with Jenkins

COURSE INTRODUCTION

- My name
- My job role
- My background in:
  - DevOps
  - Source Control Management (Git)
  - Configuration Management (Chef)
  - CI/CD (Jenkins)
- My location (for virtual classes)
- My expectations from this course

LAB: ACCESSING YOUR LAB ENVIRONMENT:



- **Local Workstation Requirements:**
  - Functioning microphone and speakers are required
  - If you wish to NOT speak at all during the course, please notify your instructor
  - At least one external monitor is needed, in addition to your computer monitor.
    - While this is not a requirement, it is strongly recommended
  - An SSH Client (PuTTY, MobaXterm, Terminal, etc.)

- **Remote Workstation:**
  - Your instructor will assign a remote workstation to you
  - You will connect from your local workstation to the remote workstation, from which you will run many of your lab exercises
  - SSH Login command:
    - Find SSH instructions for MAC/Linux or Windows here: https://bit.ly/lab-ssh

- ## Accessing Labs
  - Look for END OF LAB notation on lab docs
  - Instance IP’s
    - To be shared by your instructor

- **Github.com**
  - Create a free account if you don’t have one
    - Use a personal email address if your corporate email doesn’t allow account verification
  - Do NOT check the “Help me set up an organization next” box

- ## Artifactory server

  - http://artifactory.pipeline.builders/artifactory
  - Username: artifactory / Passwd: art123

- **Chef Server:**
  - https://manage.chef.io
  - Create a free account
  - Create an Organization
    - use lowercase characters,
    - letters, numbers and dashes are allowed

- **Chef-managed node**
  - Instructor will assign these IP Addresses to you during the Chef section of the course

SOURCE CONTROL MANAGEMENT WITH GIT

Objectives

- Configure your workstation to work with git projects
- Create a new local and remote repository
- Create branches
- Push and pull changes to and from repos

Prerequisites

- You have a verified Github account

Git purpose and use case

- Why people use git
  - Backups
  - Versioning
  - Collaboration
  - Branching
- Who uses Git
  - DevOps, Developers

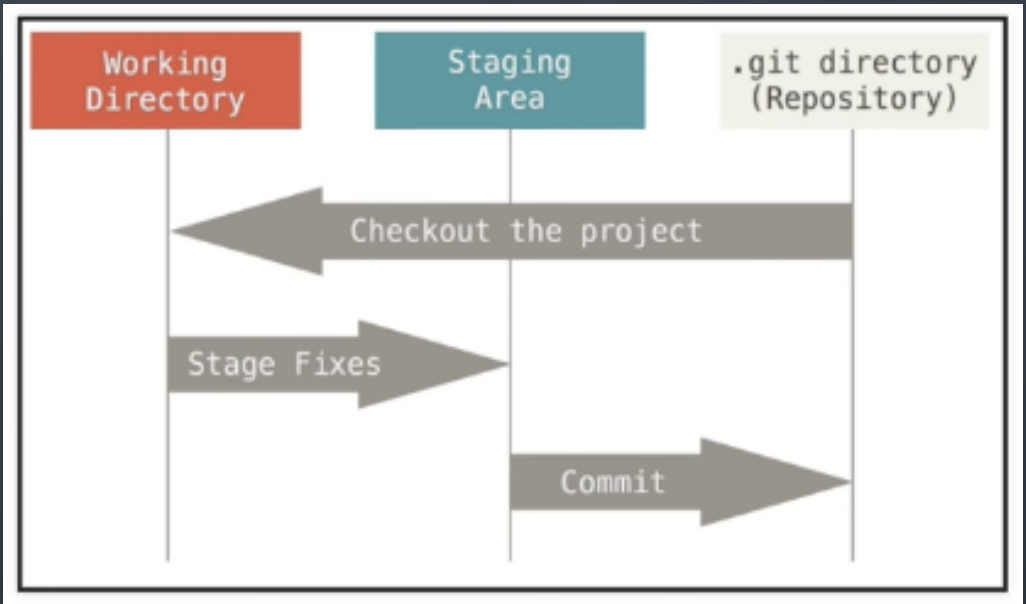


Image: Scott Chacon and Ben Straub (2014) Pro Git 2nd edition, Apress

Public versus private repositories

- Public: anyone can view and clone, but only those with permission can modify
- Private: permission required to view, clone or modify

Verifying git

- Verifying git is installed on a workstation
  - $ git --version

Group Lab - Getting help

- git help
- git help <verb>
  - git help <verb>
  - git verb --help
- Try these examples:
- $ git help add
  - ‘q’ to quit
- $git add --help
  - ‘q’ to quit

System vs. Global vs Local Configurations

- These configuration settings change how your local workstation interacts with git
- There are three scopes, in the order in which they are read:
- System: for machine you share with other people. not user based, so only for settings that are system wide
- Global: this would be used if you have one user account with multiple repos, so the same settings would affect all repos
- Local: changes only affect current repo. Good if you have other repos/accounts that you use with different settings

The last setting read wins (first system, then global, then local)

Setting user info

- $ git config --global user.name “firstname lastname”
- $ git config --global user.email “<email@address>”

- Verify setting
  - git config --global --list

Using the git editor

- Can also modify git files via an editor
  - This uses your default editor
  - To set the editor

```
$ export EDITOR=vim
$ export EDITOR=nano
$ export EDITOR=emacs
```

- To open editor:
  - $ git config --global --edit

Basic git concepts

a. Start with untracked files. Git knows they exist, but they are not being tracked by git
b. Tag the files that you want added to the repo with the 'git add' command
    i. Files included in 'git add' are now ready to be committed
    ii. $git add filename.txt (adds one specific file, named 'filename.txt')
    iii. $git add . (adds all files in that directory and all subdirectories)
c. Commit files into the repo with the 'git commit' command
d. These files can be pushed from the local repo (eg. on your workstation) to a remote repo (eg. on Github.com) using the 'git push' command

Basic git concepts

- Typical workflow:
  make changes to content -> git add -> git commit -> git push
- Branch: specific version of code within a repo
- Multiple branches representing multiple states of code:
  - Personal branch (often called a ‘feature branch’)
  - Testing
  - Staging
  - Master / Production
- Merging: taking content of one branch and moving to another branch

Pull Request

- Pull request: an offer to add code to a branch where they don’t have write permissions
- Branch owner reviews pull request and either approves integrating the offered code into the branch, or declines

Basic Git Commands

Don’t run these commands yet (we’ll do them in a later lab)

- $ git init - initializes a directory structure to be git-managed
- $ git status
  - Shows current branch
  - Shows files that have been added, but not yet committed under “Changes to be committed”
  - Shows files that have been changed but not yet added under “Untracked files”
  - Will not show committed files

Basic Git Commands (con't)

Don't run these commands yet (we'll do them in a later lab)

- $ git add - stages the untracked files (changed but not yet managed by git)
  - Files move from "Untracked" to "Changes to be committed"
- $ git commit - commits the staged files to the repo
  - Files move from "Changes to be committed" to a committed state in the repo
- $ git log - shows commit history

Commit methods

- Add a relevant message in the commit, save and quit
- Upon save and quit, your file(s) will be committed to the local repo
- Using the editor is the appropriate way to commit messages, instead of using the -m flag (eg. don’t use $git commit -m “commit message”)
  - Longer commit message gives more info, help, etc.
  - You also see other info on your screen (current branch, which files have been changed, etc)

Staging a File

```
$ git commit
```

```
Added a README file
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Committer: Ubuntu ubuntu@ip-172-31-22-72.ec2.internal
# On branch master
# Changes to be committed:
#    modified:   README
#
```

Git Log

- Shows the commit history
- Includes the SHA (Secure Hash Algorithm)
  - SHA is the unique identifier for this commit
- Shows user who made the commit

Git Show

- Shows the detail about a particular commit
- Identify the commit with the SHA

git show - Compare Differences (1 of 2)

$ git show [SHA]

```
commit 4bc4fa6ebb7dc103a700d492b09d2c9eda9f270b
Date:   Fri Jan 11 03:21:21 2019 +0000
diff --git a/README b/README
--- a/README
+++ b/README
@@ -1 +1 @@
-README v1
+README v2
```

git show - Compare Differences (2 of 2)

```
$ git show 4bc4fa6ebb7dc103a700d492b09d2c9eda9f270b
commit 4bc4fa6ebb7dc103a700d492b09d2c9eda9f270b
Date:   Fri Jan 11 03:21:21 2019 +0000
diff --git a/README b/README
--- a/README
+++ b/README
@@ -1 +1 @@
-README v1
+README v2
```

Solo Lab: Basic Git Commands

https://bit.ly/lab-basic-git

Branching

- Allows for snapshots of code changes
- Changes have to be intentionally accepted or ‘merged’ into another branch, keeping rogue changes from entering working branch of code
- Different branches for different purposes:
  - Sallys-feature-branch (which belongs to Sally, for her to experiment with)
  - Staging (for testing prior to releasing to production)
  - Master (which would usually be the “official” branch)

Branching

- Example scenario:

1. Sally perfects the code in her personal branch “Sallys-feature-branch”
2. Sally then merges her code into the ”Staging” branch for testing
3. Once her code passes the Staging tests, the owner of the Master branch can merge Sally’s changes into Master
4. The code in the Master branch would be deployed on Production servers

Branching Commands

- $ git branch
  - shows you the current branch
- $ git checkout -b <branch_name>
  - creates a new branch and then switches to that new branch
- $ git checkout <branch_name>
  - switches to a different branch

Merging Branches

- Merging branches populates one branch with the content from another branch

- Merge executed from the branch you want to merge INTO:
  $ git merge <branch_name>
  This will merge the changes from <branch_name> into the current branch

- Example: to merge ‘testing’ branch into ‘master’ branch:

  $ git checkout master (now we are on the master branch)

  $ git merge testing (merges contents of testing branch into master branch)

Deleting Branches

- Branches can be deleted with:

  $ git branch –D <branch name>

Solo Lab: Basic Git Commands

https://bit.ly/lab-git-branching

SSH Primer

a) Public versus Private keys

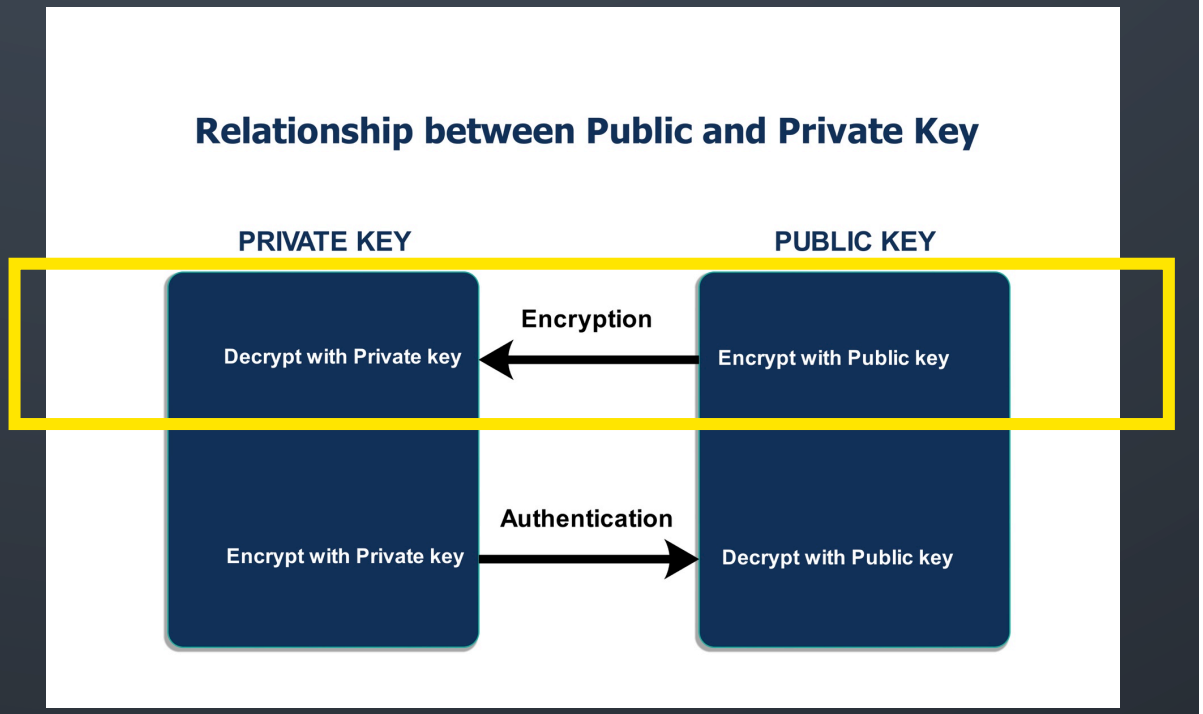
- Public and private key pair are algorithmically linked
- Typically, anyone can have your public key, and you keep your private key hidden

Relationship between PUBLIC and PRIVATE key:

Anything ENCRYPTED with the public key can only be DECRYPTED with the private key.

Allows anyone holding your public key to send an encrypted message which can ONLY be decrypted (read) by you, the holder of the private key

This is used to send ENCRYPTED messages that only you can read



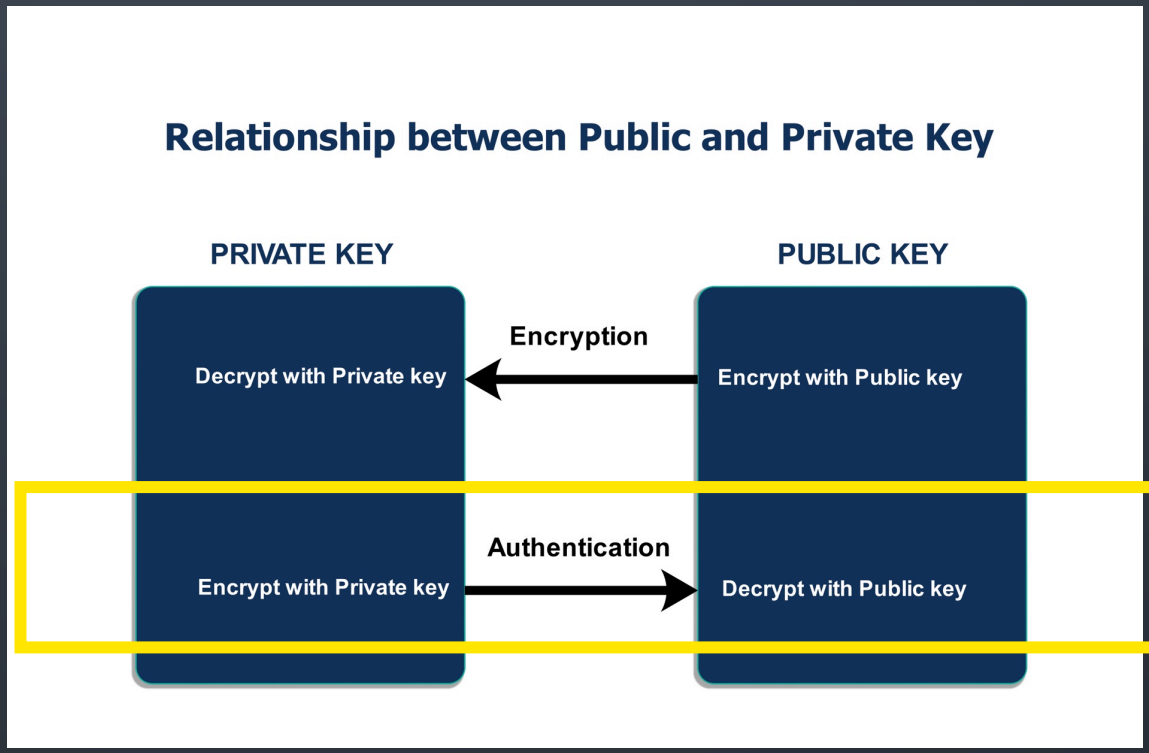
Relationship between PUBLIC and PRIVATE key:

Anything ENCRYPTED with the private key can only be DECRYPTED with the public key.

If someone can decrypt a message using your public key, then it HAD TO come from you, the holder of the private key

Since many people might have your public key, this isn't used for encryption

This AUTHENTICATION is to prove the message came from you, the holder of the private key



SSH Primer

ssh-keygen command - generates a public-private key pair

ssh-add command − adds key to your workstation keychain

Using a Remote Git Repository

- Uses a Git-compliant service such as: Github, Bitbucket, Gitlab
- Enables collaborative workflows
- Enables sharing
- Tracks changes
- Tracks bugs and issues
- Tracks contributions with stats
- Remote backup

Using a Remote Repository (2)

- Single source of truth
- Work is being done locally, so git synchronizes multiple people's changes
- Work can be done offline and synced correctly later
  - In the past, needed constant connection to sync with a remote repo

Configure Connection to a Remote Repo

- Define the remote repo, and map this connection to a keyword (usually “origin”)
  $ git remote add origin git@github.com/<username>/repo_lab.git

- If private repo, use SSH Key for authentication

- Once changes are committed locally, they can be pushed to a specific branch on the remote repo
  $ git push <remote-repo> <remote-branch>
  $ git push origin master
  $ git push origin testing

Configure Connection to a Remote Repo (con't)

- Content can be pulled from a branch on the remote repo to your local workstation
  $ git pull origin master

- Use the --set-upstream flag, or –u shortcut, to set this as the default mapping
  Going forward, only $ git push is needed

  $ git push -u origin master (sets the default values)

  $ git push (will push to the origin repo, to the master branch)

Other Useful Git Commands:

- To delete a remote branch, eg. on Github:
  $ git push origin --delete <branch-to-delete-from-github>
- To revert to an older commit:
  - Find the commit ID in the Github repo, under the 'commits' link
  - It will be a hex-based number, eg. 588d062
  - $ git reset --hard <commit-id>
    $ git reset --hard 588d062
- To update a repo after you have removed files use the −A option:
  - $ git add -A .
  - $ git commit
- To copy and overwrite all your local branches to the remote repo:
  - $ git push origin --mirror

Solo Lab: Configuring SSH for Git

https://bit.ly/lab-git-ssh