DevOps Training

Week 02:

Introduction to Version Control and Git

Muhammad Sohaib Ali

/msohaibali

Week 1 Recap

- Introduction to DevOps
- Intro to Virtual Machines
- Setting up **Ubuntu** based VM on **Digital Ocean**
- Basic **LINUX** commands

Before Moving Forward

- Any **Questions/Queries** from Previous Week?

Week 2 Coverage

- Missed Concepts related to VMs
- Some useful **LINUX** commands
- Introduction to **Version Control**
- Setting up **Git**
- Overview and hands on **GitHub**

Virtualization

Why learn Virtualization?

- Modern computing is **more efficient** due to virtualization
- Virtualization can be used for mobile, personal and cloud computing
- If you understand virtualization, you will be able to understand why **containers** exist in first place
- Virtualization is the **backbone** of modern computing infrastructure
- As a **DevOps Engineer**, you will have to deal with it on daily basis

What is a Hypervisor?

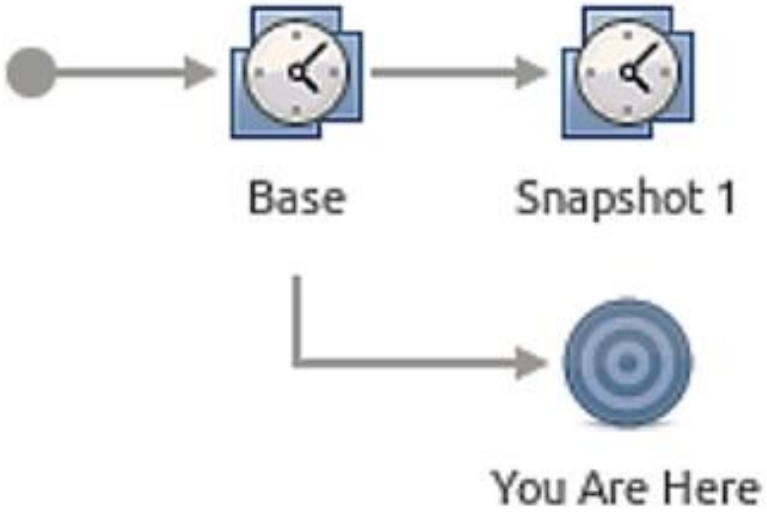
- Software installed **on top of** hardware that created virtualization layer.
- A hypervisor, also known as a virtual machine monitor (**VMM**), is a software layer that enables multiple operating systems (OS) to **share a single physical host machine** by abstracting its hardware resources, such as CPU, memory, storage, and networking, into virtual resources that can be allocated dynamically to virtual machines (VMs).

Virtual Machine Files

- VMs can be **Exported** and moved to other hosts
- Files are created by the hypervisor and stored in a **directory**
- They are **stored** in specific format and can be shared, copied, moved to another computer

What is a Snapshot?

- Working on a VM and need to **save progress** or state
- Snapshots are saved as files in the VM folder (<vmname>.vmx)
- What is saved by a snapshot?
  - ❖ State of VM disks
  - ❖ Contents of VM memory
  - ❖ VM settings



Why should we use virtual machines?

- **Full Autonomy**: it works like a separate computer system; it is like running a computer within a computer.
- **Very Secure**: the software inside the virtual machine cannot affect the actual computer.
- **Lower Costs**: buy one machine and run multiple operating systems.
- Used by all Cloud providers for **on-demand** server instances.
- Softwares used for Virtualization
  - **VirtualBox**
  - **VMWare**
  - **Parallels**

Limitations of use virtual machines?

- Uses **hardware** in Local Machine
- Not very **Portable** since size of VMs are large
- There is an **Overhead** associated with virtual machines (dedicated resources)
- Guest is not as **fast** as the host system
- Takes a **long time** to Start up
- It may not have the same **graphics capabilities**

Linux

Useful Commands

Commands: Hands-On

After you connect, type

- **whoami** # my login
- **hostname** # name of this computer
- **echo** “Hello, world” # print characters to screen
- **echo $HOME** # print environment variable
- echo my login is $(whoami) # replace $(xx) with program output
- **date** # print current time/date
- **cal** # print this month’s calendar

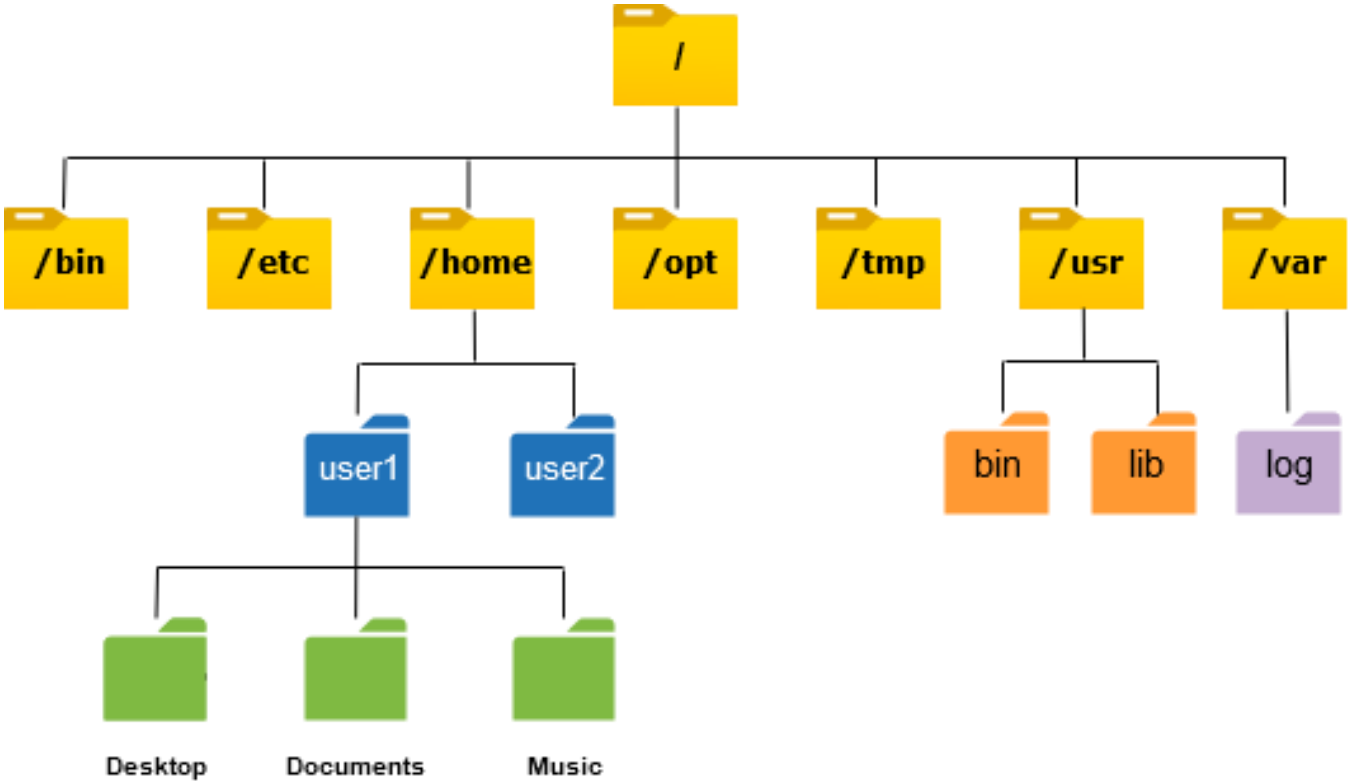
Working with shell

Ubuntu Desktop
/ Graphical View

Linux Shell

```
$ echo  Hello
Hello
$
```

Linux Directory Structure



Command Types

Shell built-in commands: These are the commands that are built into the shell itself, such as cd, echo, and pwd.

System commands: These are the commands that interact directly with the operating system, such as ls, cp, and rm.

External commands: These are the commands that are not part of the shell or the operating system, but are instead separate programs that can be executed from the command line, such as grep, gedit,etc.

ls: List files and directories in the current directory

Syntax: ls [options] [directory]

Example: ls lists all files and directories in the current directory

pwd: Print the current working directory

Syntax: pwd

Example: pwd prints the current working directory

mkdir: Create a new directory

Syntax: mkdir [options] directory_name

Example: mkdir my_folder creates a new directory called "my_folder"

mkdir ls: Create a new directory and list its contents

Syntax: mkdir [options] directory_name && ls [options] [directory_name]

Example: mkdir my_folder && ls -l creates a new directory called "my_folder" and lists its contents in long format

cd: Change the current working directory

Syntax: cd [directory_path]

Example: cd /home/user/Documents changes the current working directory to "/home/user/Documents"

mv: Move or rename a file or directory

Syntax: mv [options] source destination

Example: mv file.txt /home/user/Documents moves the file "file.txt" to the directory "/home/user/Documents"

cp: Copy a file or directory

Syntax: cp [options] source destination

Example: cp file.txt /home/user/Documents copies the file "file.txt" to the directory "/home/user/Documents"

cp -r: Copy a directory and its contents recursively

Syntax: cp -r [options] source destination

Example: cp -r my_folder /home/user/Documents copies the directory "my_folder" and all its contents recursively to the directory "/home/user/Documents"

Absolute and Relative Path

Absolute path: An absolute path specifies the complete path to a file or directory from the root directory. It starts with a forward slash (/) and shows the entire directory hierarchy starting from the root directory.

Example: Suppose you have a file called "file.txt" located at /home/user/Documents/. To access it using an absolute path, you would use the entire path from the root directory, like this: /home/user/Documents/file.txt.

Absolute and Relative Path

Relative path: A relative path specifies the path to a file or directory relative to the current working directory. It does not start with a forward slash (/) and shows the directory hierarchy starting from the current working directory.

Example: Suppose you are currently located in the directory /home/user/ and you want to access the file "file.txt" located in directory /home/user/Documents/. You can access it using a relative path by specifying the path relative to the current working directory. In this case, relative path would be Documents/file.txt.

cat: Concatenate and display files

Syntax: cat [options] file_name

Example: cat file.txt displays the contents of the file "file.txt"

touch: Create an empty file or update the timestamp of an existing file

Syntax: touch [options] file_name

Example: touch file.txt creates an empty file called "file.txt" or updates the timestamp of an existing file

less: View the contents of a file one page at a time with more features than more

Syntax: less [options] file_name

Example: less file.txt displays the contents of the file "file.txt" one page at a time. You can navigate through the file using the arrow keys or page up/down, search for text using /, and quit using q.

Getting help in Linux command line

man command: The man command displays the manual pages for a specific command. The manual pages contain detailed information about the command, including its syntax, options, and usage examples. To use the man command, simply type man followed by the name of the command you want help with.

Example: man ls displays the manual pages for the ls command.

Getting help in Linux command line

--help option: Many Linux commands also have a --help option that displays a short help message. This option usually provides a brief description of the command, its syntax, and its most common options.

Example: ls --help displays a short help message for the ls command.

Sudo

sudo is a command in Linux that allows users to run commands with administrative privileges.

Examples:

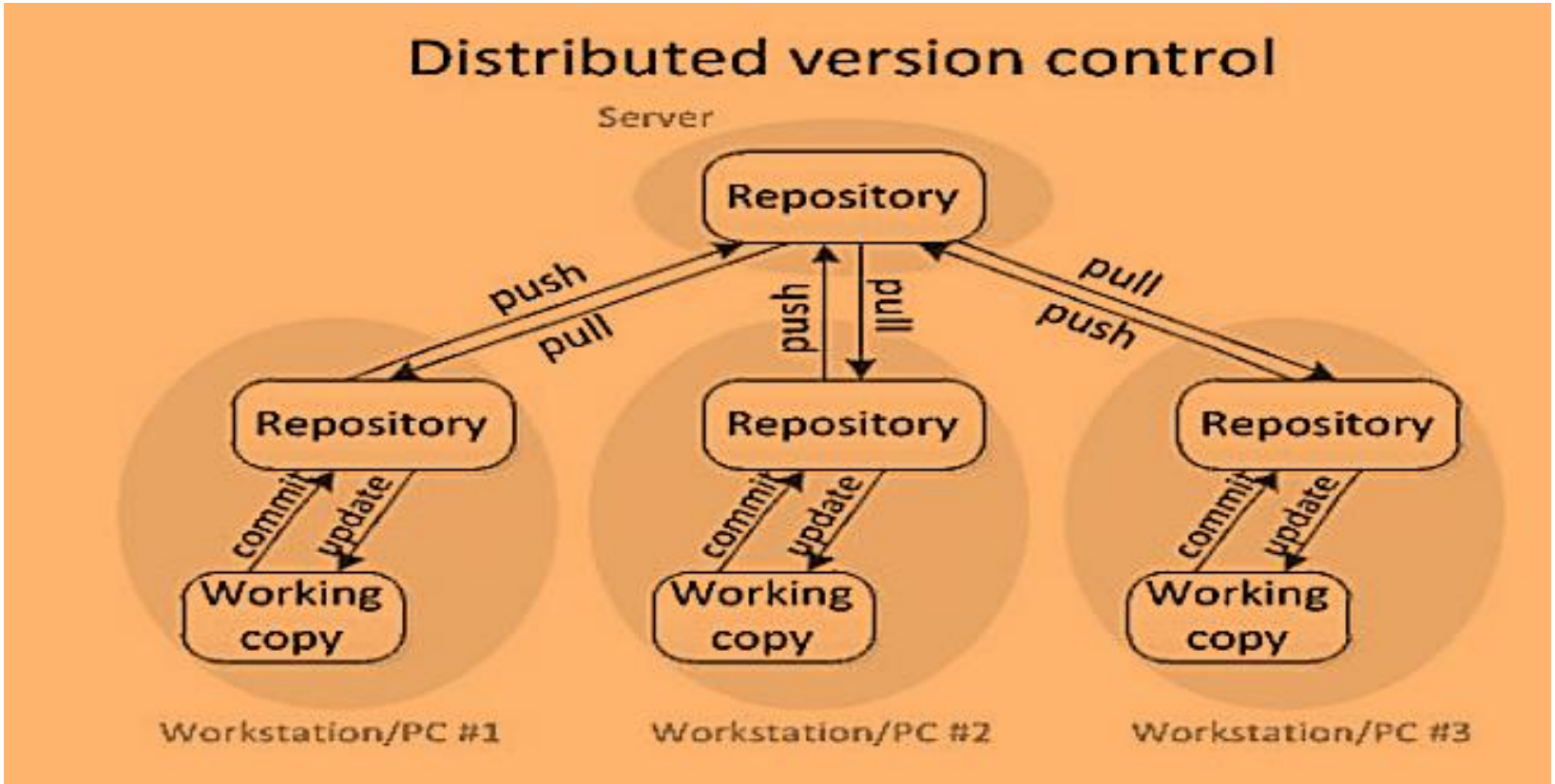
- Installing or updating software
- Modifying system settings or files
- Restarting services
- Adding or removing users
- Running commands that require root-level access

Version Control Systems

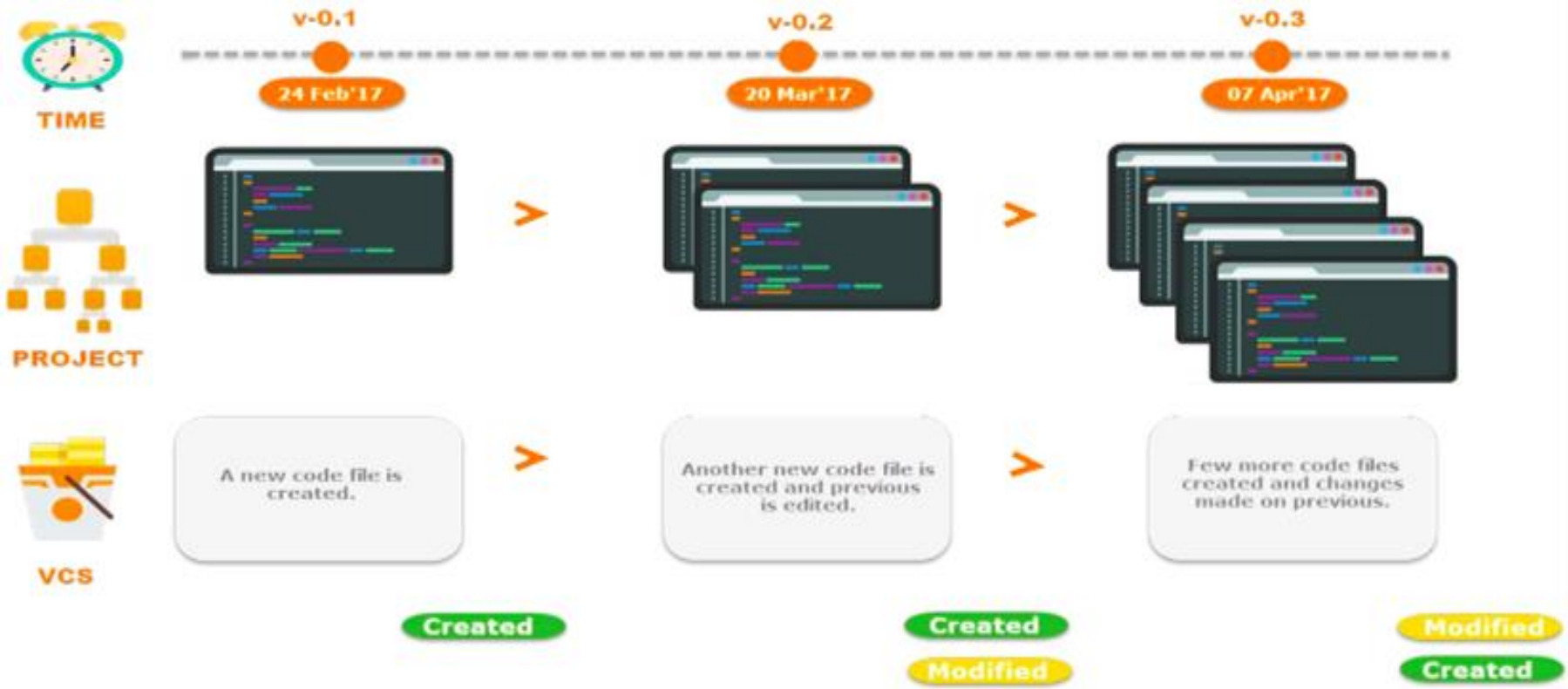
What is Version Control System (VCS)?

- Version Control System (VCS) is a software tool for managing **changes** to files and folders over time.
- VCS allows you to **track** every **modification** made to files and folders, creating a **history** of changes.
- You can **compare different versions** of files and folders, and **roll back** to previous versions if needed.
- VCS enables **collaboration** on the same files and folders, making it easier to work with others and **avoid conflicts**.
- VCS is commonly used in software development but can be helpful for **managing** any type of **digital content** that undergoes **changes over time**.

What is Version Control System?



What is Version Control System?



Tools for Version Control

Examples of VCS tools include

- Git
- SVN
- Mercurial
- Perforce

GIT

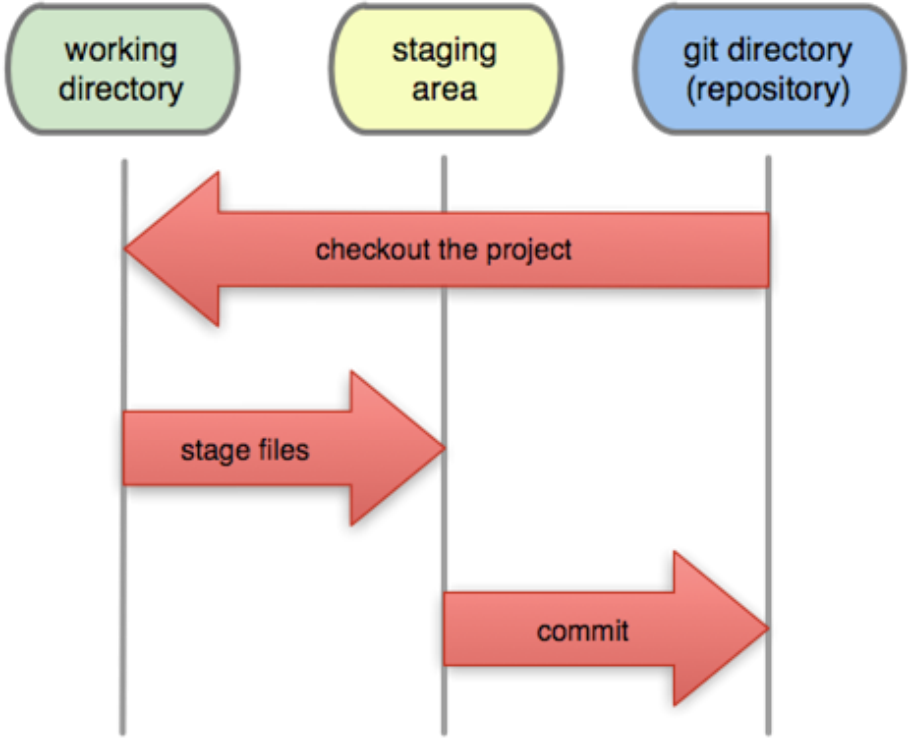
Git

- Git is a **DVCS** for Small to Large projects
- Tracks **changes** in files/folders in a local repository
- Enables easy **reversion** to previous versions
- Allows **collaboration** among multiple developers
- Uses branches for **simultaneous** work on different versions
- **Merges** changes back into the main branch
- Popular among software developers

Feature	Git	GitHub	GitLab	Bitbucket
Type of Tool	Distributed Version Control System	Web-based Git repository hosting service	Git repository management platform	Git repository management platform
Ownership	Developed by Linus Torvalds	Owned by Microsoft	Owned by GitLab Inc.	Owned by Atlassian
Hosting	Local, Self-hosted or third-party	Hosted, Owned and managed by GitHub	Hosted, Owned and managed by GitLab	Hosted, Owned and managed by Atlassian
Pricing Model	Free and Open Source	Free for public repositories, paid for private repositories	Free and Open Source, with premium options	Free for up to 5 users, paid for larger teams
Features	Basic version control, branching and merging	Bug tracking, project management, code review, wiki	Continuous Integration, DevOps, security scanning	Integration with Jira, project management, code review, wiki
Popularity	Widely used among developers	Most popular Git hosting service	Growing popularity among developers	Popular among software development teams
Languages	Supports all programming languages	Supports all programming languages	Supports all programming languages	Supports all programming languages

Stages in Git

- Git has three main stages:
  - the working directory
  - the staging area
  - the repository



Install Git

- Ubuntu: $ sudo apt-get install git-all
- OR
- Install Git: https://git-scm.com/downloads

Git Repository

- A single project containing the code base of your application/service Can be
  - **Private**: Only certain people with access can see the repo
  - **Public**: Anyone can see the repo
- In your local git repository, a **.git** folder is created, it basically tracks all the changes/versions of your codebase.
- There is a local repository & a remote repository, you make changes to local repository and push them to remote.

CREATING GITHUB ACCOUNT

- Go to github.com and Signup, your username will be your github address
- E.g. github.com/msohaibali

CONFIGURE GIT

- Configure your git with your config so that your commits can have this configuration.
  - git config --global user.name "Your Name"
  - git config --global user.email "you@example.com"

- For pushing changes to Github/Cloning a private repo, it needs to authenticate you, so everytime, you will have to enter your username/pwd.

Creating Git Repo

- Click the + at the top right, New Repository, Enter Repository Name, Description, Chose Public and initialize with a README.
- Your Repository will be created with following link github.com/<username>/<repo-name>

LAB - GIT REPOSITORY

- Creating Github Account
- Create & Cloning a Repository
- Add SSH key or Personal Access Token (PAT) token

Git Commands

Command	Description	Example
git status	Shows the changes made in the local repository only.	git status
git diff	Shows the differences between the files in the working directory and the ones in the staging area.	git diff <file>
git add <file>	Adds a file to the staging area.	git add <file>
git reset <file>	Removes a file from the staging area.	git reset <file>
git commit	Commits changes from the staging area to the local repository.	git commit -m "commit message"
git push	Pushes local repository changes to a remote repository.	git push origin main (pushes the main branch to the remote repository named origin)

Task

- Create Github Account
- Create Github Repository
- Clone it
- Make changes
- Check different stages of git
- Check differences in files

Git Lab

- You should have cloned the repo created in previous Lab.
  - cd <repo-name>
  - Ls
- You will be seeing the README.md le. So We will add a new file in the repo.
  - touch a.txt
- Edit this file either in vim or any text editor. Now see the status of your repo.
  - git status
- It will show a file is added that is currently untracked i.e. it is not present in git history

Git Lab

- Now add to Staging Area
  - git add a.txt
  - git status
- It means file is in staging, but it should be committed to local repo.
- Now edit the README.md file, add a bit description like "This is a test repo" or anything else.
  - git status
- It shows that new file a.txt is yet to be committed, and README.md has been modified but it needs to be staged for commit first.

Git Lab

- Now see the differences
  - git diff
  - diff --git a/README.md b/README.md index 5297f8f..2b26b81 100644
  - --- a/README.md
  - +++ b/README.md
  - @@ -1 +1,2 @@
  - -# test-git
  - \ No newline at end of file
  - +# test-git
  - +This is a test repo
- It shows that -# test-git was removed and +# test-git, +This is a test repo were added.

Git Lab

- Add this file to Staging and then see status
  - git add README.md
  - git status
  - On branch master
  - Your branch is up to date with 'origin/master'.
  - Changes to be committed:
  - (use "git reset HEAD <file>..." to unstage)
  - modified: README.md new file: a.txt
- Check difference in staging area
  - git diff --staged

Git Lab

- Commit Change to Local Repo
  - git commit -m "Add a.txt and update README"
  - git status
  - On branch main
  - Your branch is ahead of 'origin/main by 1 commit.
  - (use "git push" to publish your local commits) nothing to commit, working tree clean
- It shows that your local branch is ahead of origin/main by 1 commit that we just did. And the working tree is clean i.e. no other file has been changed or is in staging area.

GIT: PUSH & PULL

Push:

- Transfer commits from your local repository to a remote repository
- Share modifications with remote team members
- Commands
  - git push origin

Pull:

- Download content from a remote repository and update the local repository
- Get modification from remote team members
- Commands:
  - git pull origin

LAB: GIT PUSH

- Edit a file or create a new file
- Add to staging area
- Create a Commit to Local Repo
- Push:
    - git push origin main

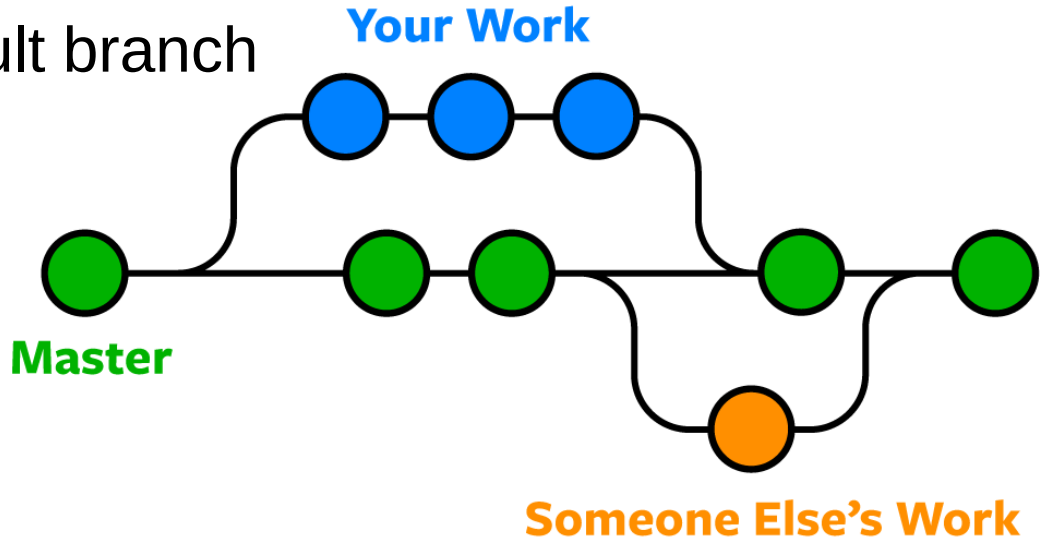
LAB: GIT PULL

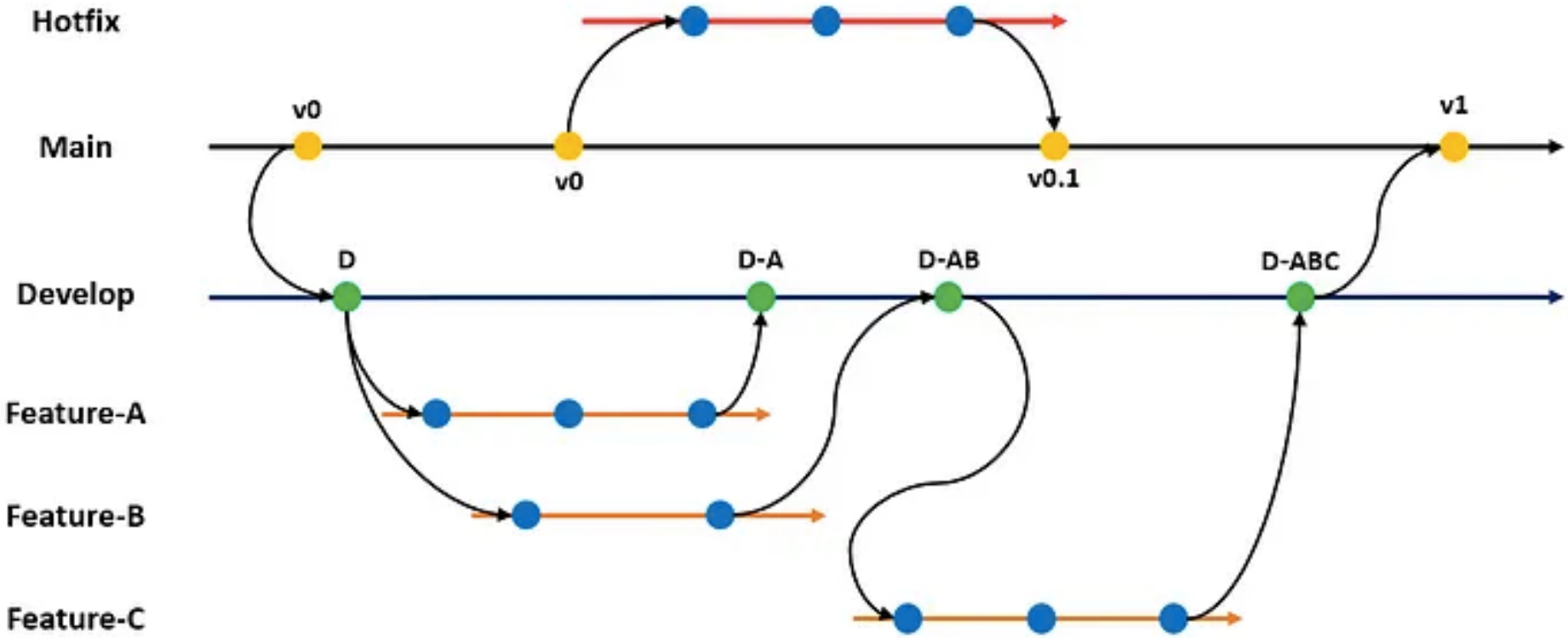
- Go to github.com UI and create a new file from UI & Commit from UI
- Check the Remote Commit
- Pull:
    - git pull origin main

List contents of your local repository, you will see the newly created
file

Branching Basics

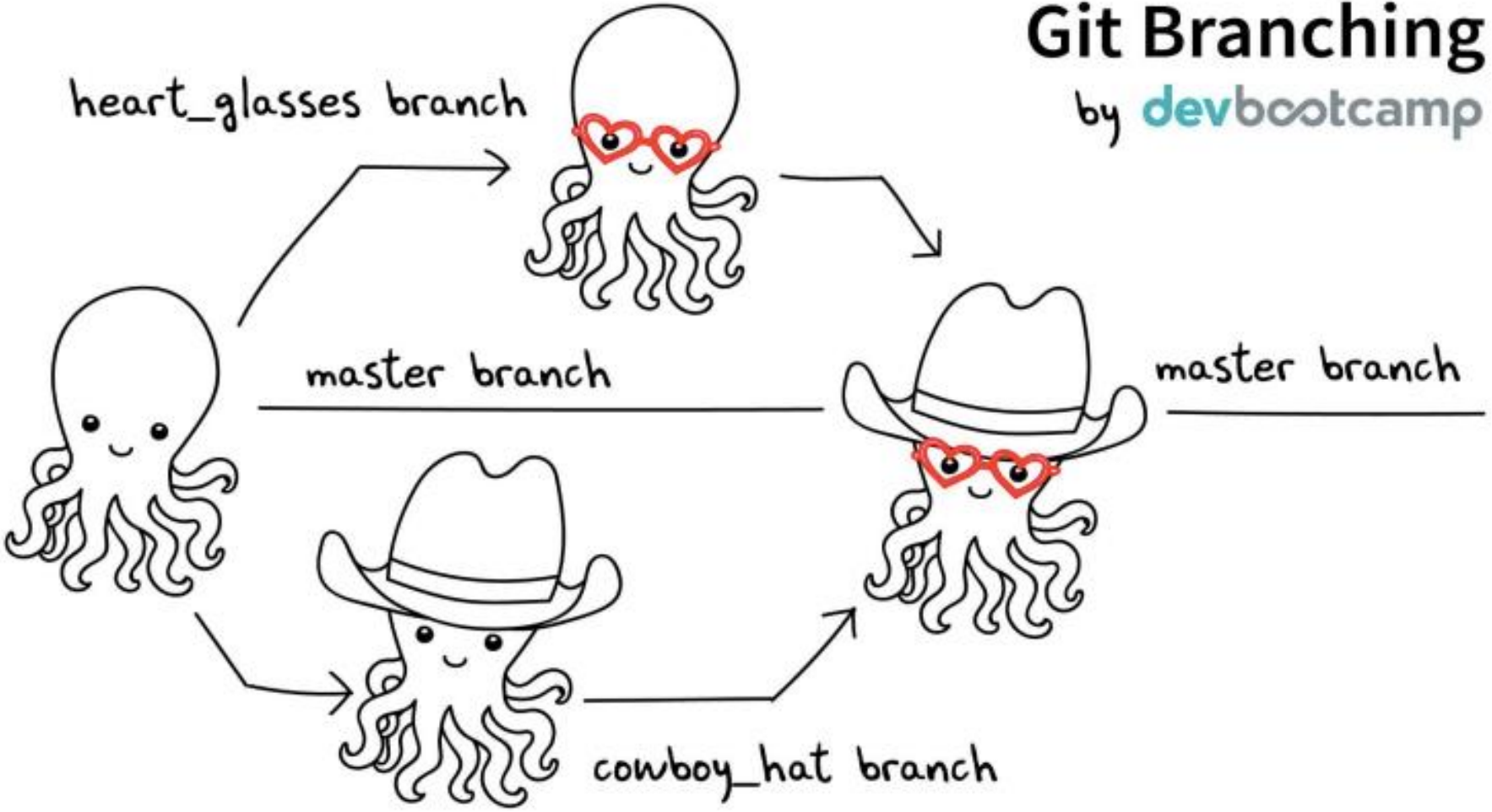
- Independent/isolated line of development
- Inexpensive/lightweight as compared to other VCS
- Way to work on different version of a repository
- Work in parallel
- Main/Master is the default branch




Hotfix
Main
v0
v0
v0.1
v1
Develop
D
D-A
D-AB
D-ABC
Feature-A
Feature-B
Feature-C

Git Branching

by devbootcamp

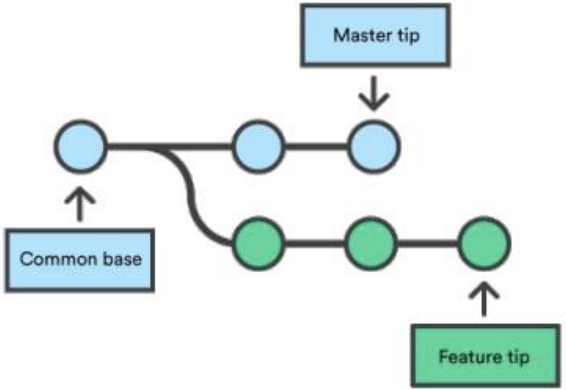


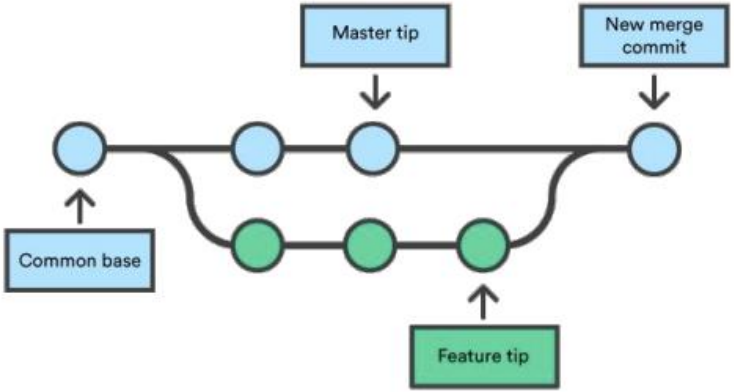
Git branch-related commands

Command	Description
git branch	List all local branches.
git branch -a	List all local and remote branches.
git branch [branch_name]	Create a new branch with the given name.
git checkout [branch_name]	Switch to the specified branch.
git branch -d [branch_name]	Delete the specified local branch.
git push origin --delete [branch_name]	Delete the specified remote branch.
git checkout -b [branch_name]	Create a new branch with the given name and switch to it.

Merging

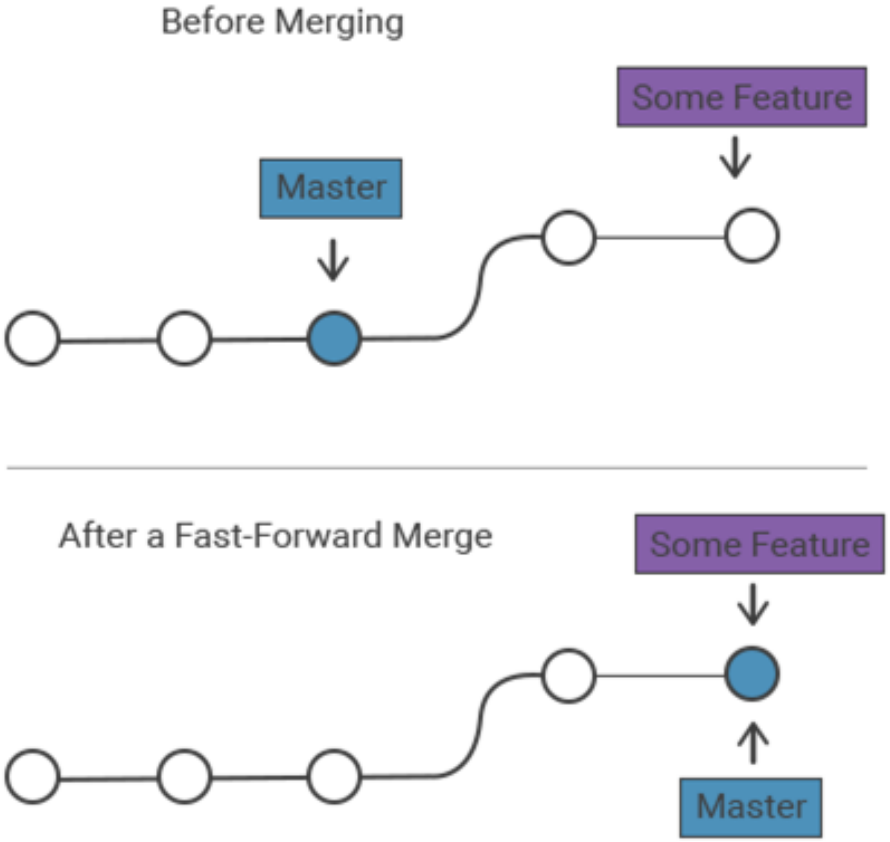
- Combine multiple branches into one
- Combines multiple sequence into unified history
- Commands:
  - Merge branch into active branch: git merge [branch name]
  - Merge branch into target branch: git merge [source branch] [target branch]





Merging - Fast Forward

- Fast forwarding is a type of merge that occurs when the target branch of the merge can be updated to include all the changes in the source branch without creating a new commit



Lab - Branching

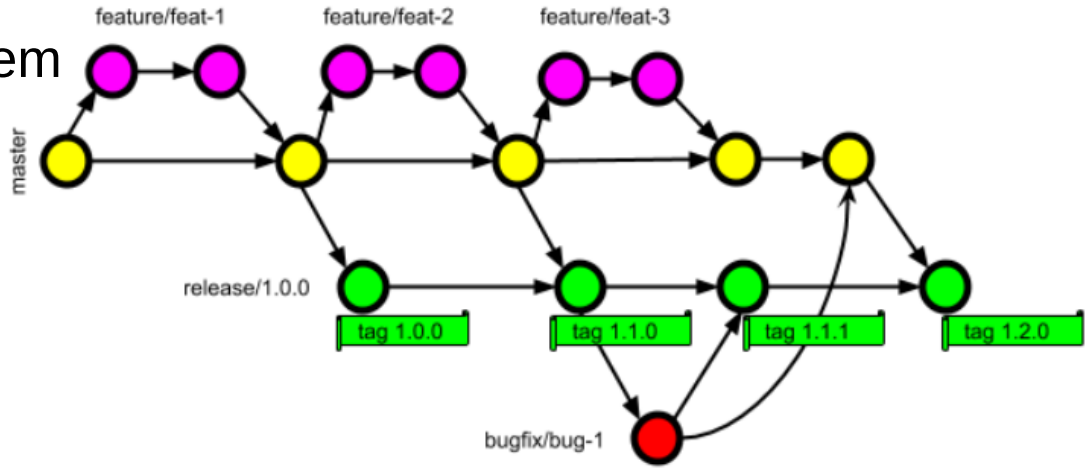
- Create Branch
  - git branch test-branch
- List Branches
  - git branch
  - git branch --list
  - Remote: git branch -r
  - All: git branch -a
- Switch Branch
  - git checkout test-branch

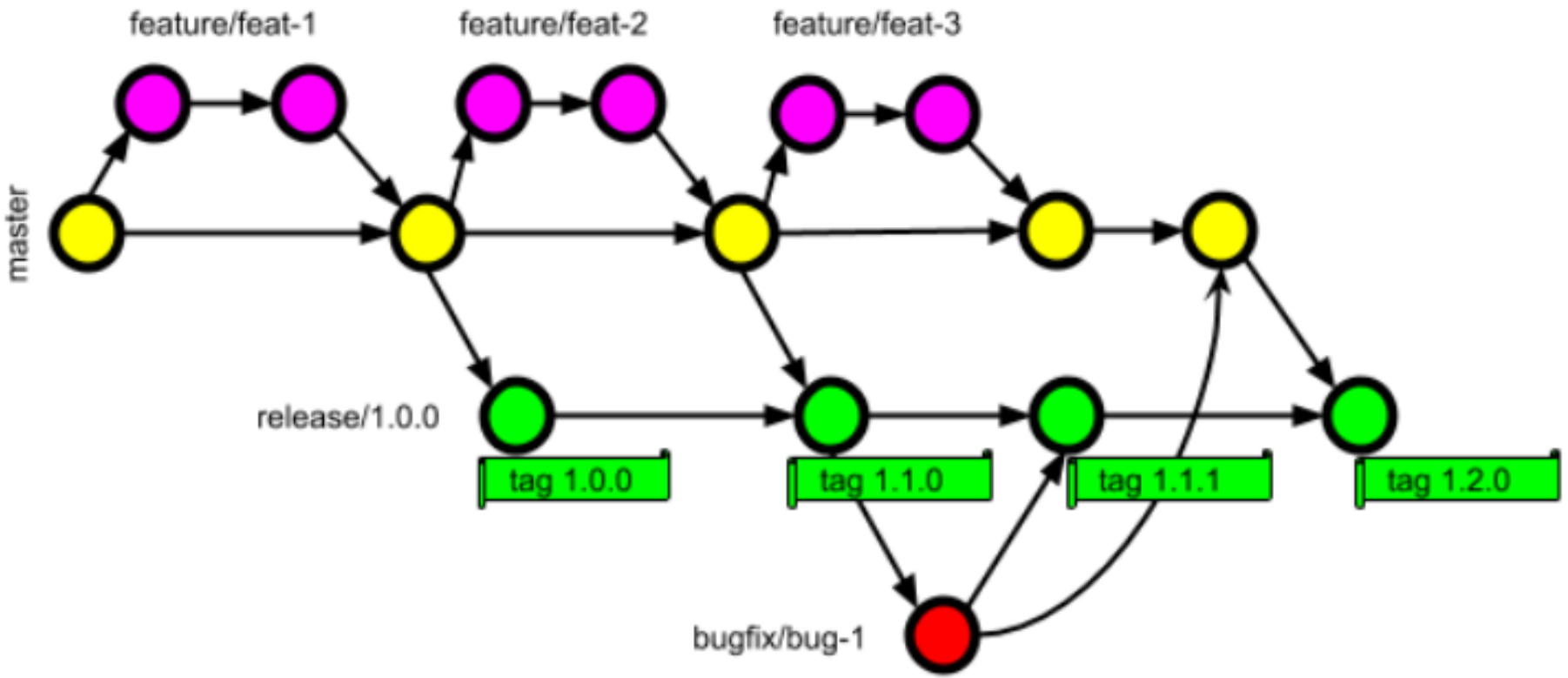
Task

- Commit to new Branch
- Switch to new branch & create a new file test-branch.txt and edit contents
  - nano test-branch.txt
  - git add test-branch.txt
  - git commit -m "create test-branch.txt file"
- **Rename Branch**
  - git branch -m test-branch test-new-branch
- **Delete Branch**
  - git checkout main
  - git branch -d test-new-branch

Tagging

- Allows you to give commit a name
- Mark important checkpoint in the project
- Tags
  - Annotated: extra metadata e.g. tagger name, email, and date.
  - Lightweight: only tag name
- Branch that doesn’t change
  - You can checkout to them




feature/feat-1
feature/feat-2
feature/feat-3
master
release/1.0.0
tag 1.0.0
tag 1.1.0
tag 1.1.1
tag 1.2.0
bugfix/bug-1

Lab - Tagging

- Create Lightweight Tag
  - git tag "first-tag"
- Create Annotated Tag
  - git tag -a "second-tag" -m "tag-test"
- List Tags
  - git tag -l
- Tag Old Commit
  - Show list of last three commits
  - git log -n3
  - Copy Commit Hash
  - git tag "third-tag" HASH

Lab - Tagging

- Get Information On a Tag
  - git show first-tag
  - git show second-tag
- Delete Tags
  - git tag --delete first-tag
  - git tag --delete second-tag
  - git tag --delete third-tag

The End