



UNIVERSITY OF  
CALGARY

# ENSF 400 - Software Engineering Industry Practices and Communication

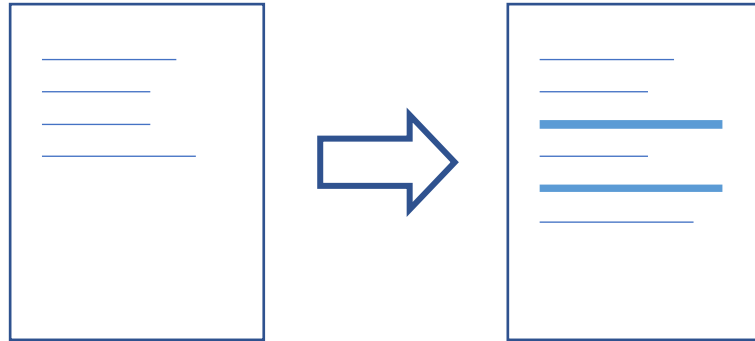
Instructors: L01 - Steve Drew; L02 - Mainul Polash; L03 - Ann Barcomb

Jan 23, 2024

Lecture 3 – Source Code Management System and Virtualization

# Why Do We Need A SCM?

- SCMs Track File Changes



- Code is organized within a **repository**.

- SCMs Tell Us:
  - **Who** made the change?
    - So you know whom to blame
  - **What** has changed (added, removed, moved)?
    - Changes within a file
    - Addition, removal, or moving of files/directories
  - **Where** is the change applied?
    - Not just which file, but which version or branch
  - **When** was the change made?
    - Timestamp
  - **Why** was the change made?
    - Commit messages

# Brief History Of SCM

- **First Generation – Local Only**

- SCCS – 1972
  - Only option for a LONG time
- RCS – 1982
  - For comparison with SCCS, see this [1992 forum link](#)

- **Second Generation – Centralized**

- CVS – 1986
  - Basically a front end for RCS
- SVN – 2000
  - Tried to be a successor to CVS
- Perforce – 1995
  - Proprietary, but very popular for a long time

- **Second Generation (Cont.)**

- Team Foundation Server – 2005
  - Microsoft product, proprietary
  - Good Visual Studio integration

- **Third Generation – Decentralized**

- BitKeeper – 2000
- GNU Bazaar – 2005
  - Canonical/Ubuntu
- Mercurial – 2005
- Git – 2005
- Team Foundation Server – 2013

# Git – The Stupid Content Tracker

- Started by Linus Torvalds – 2005
  - After the BitKeeper cannot be used for free anymore
- Efficient for large projects
  - E.g., Linux
  - Was an order of magnitude faster than other contemporary solutions
- Non-linear development!
  - Branching is cheap and easy!!!!
- [Official Site](#), [Wikipedia](#),
- [Initial README commit](#) (some profanity)

```
GIT(1)                                Git Manual

NAME
    git - the stupid content tracker

SYNOPSIS
    git [--version] [--help] [-C <path>] [-
    --exec-path[=<path>]] [--html-path
    [-p|--paginate|--no-pager] [--no-re
    --git-dir=<path>] [--work-tree=<pa
    <command> [<args>]

DESCRIPTION
    Git is a fast, scalable, distributed re
    an unusually rich command set that prov
    operations and full access to internals
```

# Terminology

- Branch

- A history of successive changes to code
- A new branch may be created at any time, from any existing commit
- May represent versions of code
  - Version 1.x, 2.x, 3.x, etc.
- May Represent small bugfixes/feature development
- Branches are cheap
  - Fast switching
  - Easy to “merge” into other branches
  - Easy to create, easy to destroy
- See [this guide](#) for best practices

- Commit

- Set of changes to a repository's files
- More on this later

- Tag

- Represents a single commit
- Often human-friendly
  - Version Numbers

- A Repository may be created by:

- Cloning an existing one ([git clone](#))
- Creating a new one locally ([git init](#))

# What is a Commit?

- Specific snapshot within the development tree
- Collection of changes applied to a project's files
  - Text changes, File and Directory addition/removal, chmod
- Metadata about the change
- Identified by a [SHA-1 Hash](#)
  - Can be shortened to approx. 6 characters for CLI use
    - (e.g., “`git show 5b16a5`”)
  - [HEAD](#) – most recent commit
  - [ORIG\\_HEAD](#) – after a merge, the previous [HEAD](#)
  - `<commit>~n` – the `n`th commit before `<commit>`
    - e.g., `5b16a5~2` or `HEAD~5`
  - `main@{01-Jan-2018}` – last commit on [main](#) branch before January 1, 2018

```
commit d6424449ced0e33af1c2b89e35ed40e2c00a29d1
```

```
Author: Nathan Grebowiec <njgreb@gmail.com>
```

```
Date: Fri Sep 19 06:50:41 2014 -0500
```

```
use querySelectorAll() in all cases
```

```
since we aren't using the HTML LiveCollection returned by  
getElementsByTagName() there is no reason to not just use  
querySelectorAll() in all cases.
```

```
commit 5b16a579a2e7c052f56f867623c301eda762fab1
```

```
Author: John Heroy <johnheroy@gmail.com>
```

```
Date: Thu Sep 18 22:01:31 2014 -0700
```

```
Remove type=text/javascript in example <script> tags
```

```
commit 58e11bd4d899fd9943231b55424a954e6398f7a3
```

```
Author: Jose Joaquin Merino <jomerinog@gmail.com>
```

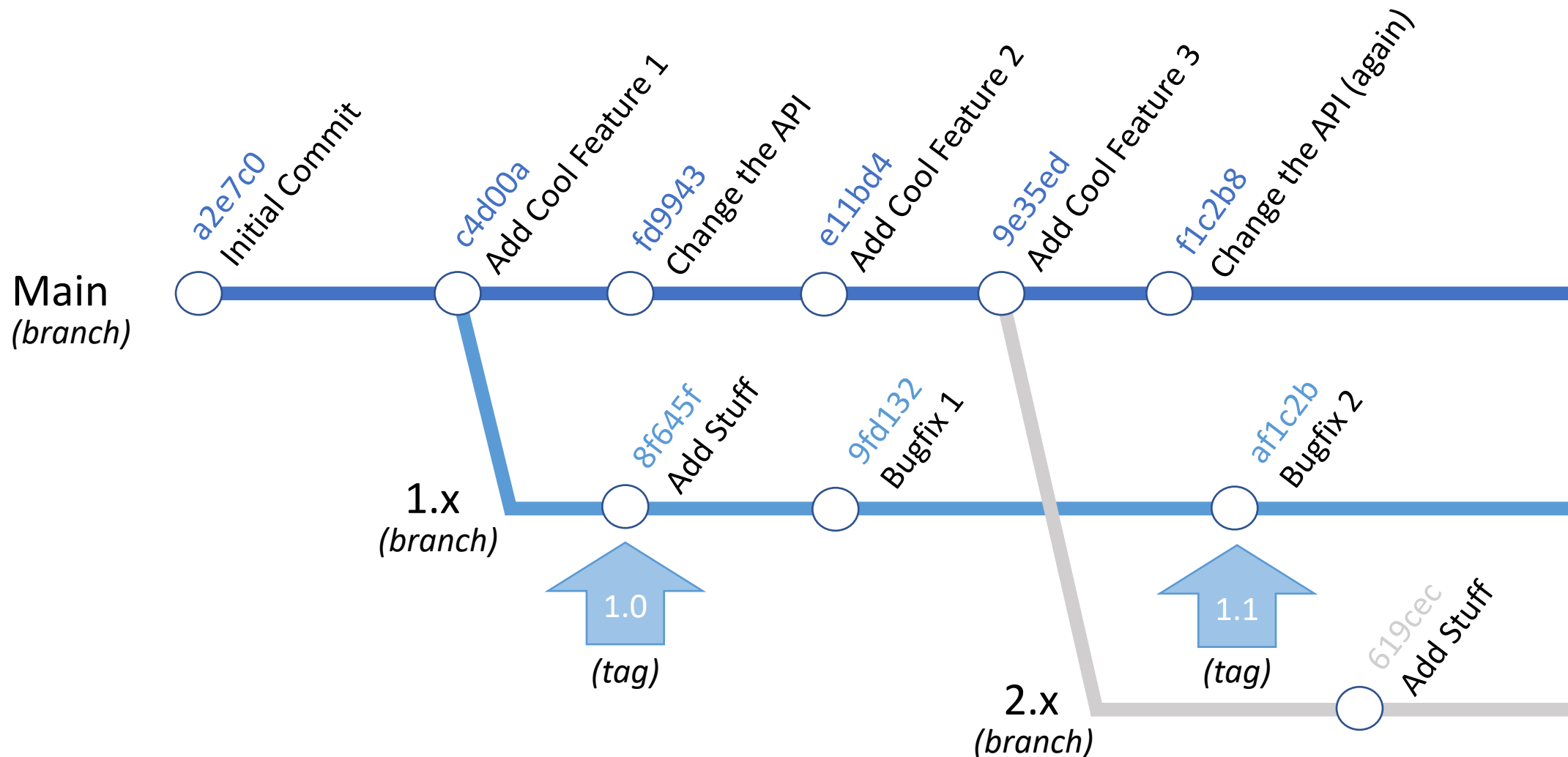
```
Date: Thu Sep 18 21:18:44 2014 -0700
```

```
Fix capitalisation and specificity of parameters
```

```
Layout changes:
```

```
- randLoadIncrement makes references to previous 'load increme  
out former after the latter.
```

# Branches, Commits, and Tags



# Terminology

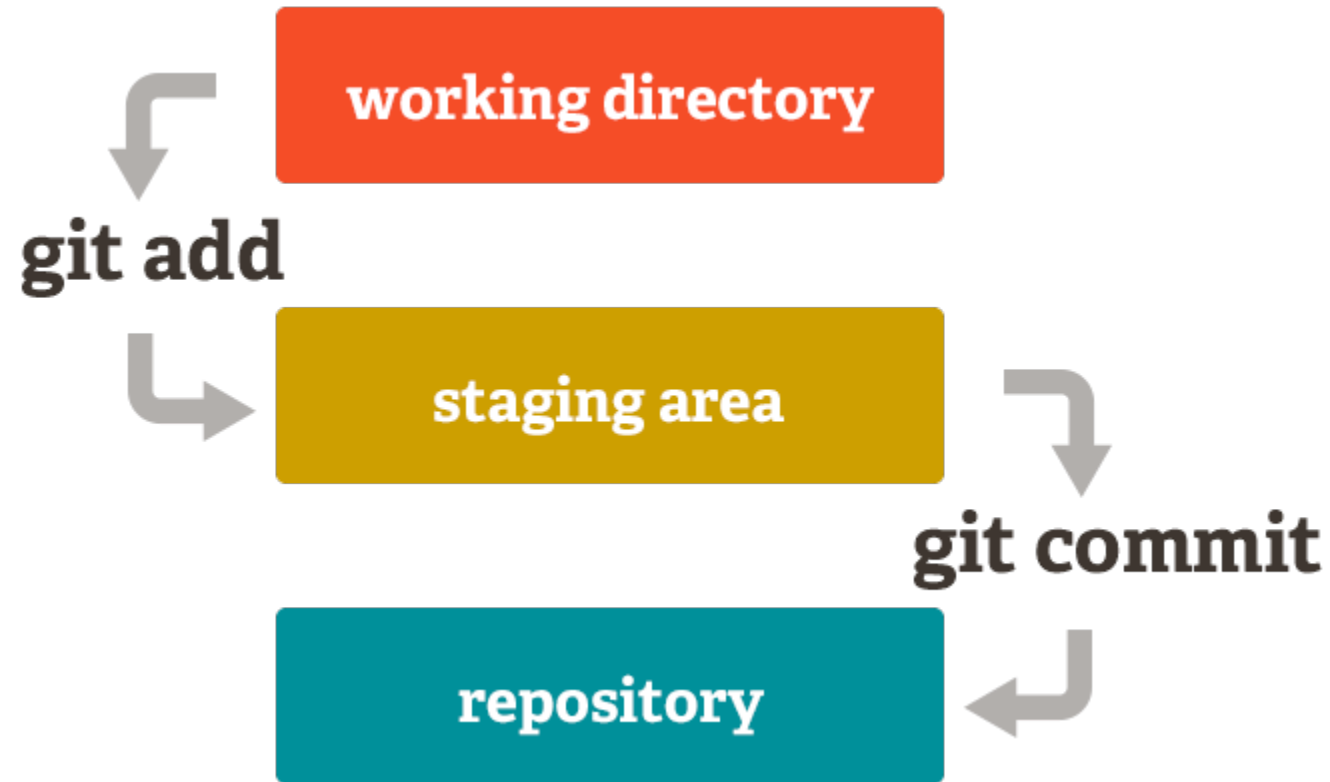
- **Working Files**
  - Files that are currently on your File System
- **The Stage** (also called the “index”)
  - Staging is the first step to creating a commit
  - The stage is what you use to tell Git which changes to include in the commit
  - File changes must be “added” to the stage explicitly
  - Only changes that have been staged will be committed
- **Checkout**
  - Replace the current working files with those from a specific branch or commit
- Use “git diff” to see which changes exist.
- Use “git add” to tell Git that a file’s changes should be added to the stage.
- Use “git status” to see the changes in your working files, as well as what changes have been staged for the commit.
- Use “git commit” to convert all staged changes into a commit.
  - git commit -m “my commit message”
  - git commit -m “my commit message” -a
    - Will automatically stage all files tracked by the repo & add them to the commit.
    - **Please don’t do this unless all commits are related.**



# A Distributed SCM

- What do we mean by **distributed**?
  - **Cloning** a repository (repo) creates a full copy of that repo on your local machine.
  - You can **fetch** updates from non-local repositories (e.g., repos on GitHub)
- A non-local repository is a **remote**
  - When you **clone** a repository, a remote called **origin** is created for you automatically in your local repo which points to the source repo.
  - Remotes are local settings. They do not become part of your commits.
  - You may set up additional remotes.
    - Use case: Development teams, where each dev has their own repository.
- **Common Pattern:**
  - **One** canonical repository for the project (on GitHub, for example)
  - Every developer **forks** that repo on GitHub
    - A **fork** is simply creating a copy of the repo on that remote system
  - Every developer **clones** their own GitHub repo to their local machine
  - Every developer adds a **remote** pointing to the canonical (main, authoritative) repo, as well as other dev repos as needed
  - Devs work **locally**, push changes to their own remote, then request that the canonical repo accept their changes via a **pull request**.

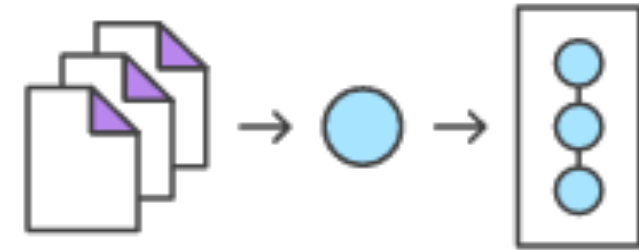
# Basic Git model locally



# Setting up a repository

- Initializing a new Git repo
  - `git init`
- Cloning an existing Git repo
  - `git clone`
- Committing a modified version of a file to the repo
  - `git add` and `git commit`
- Configuring a Git repo for remote collaboration
  - `git push`
  - `git remote add <remote_name> <remote_repo_url>`
  - `git push -u <remote_name> <local_branch_name>`
- Common Git version control command
  - `git config --global user.name <name>`
  - `git config --local user.email <email>`

# Saving changes

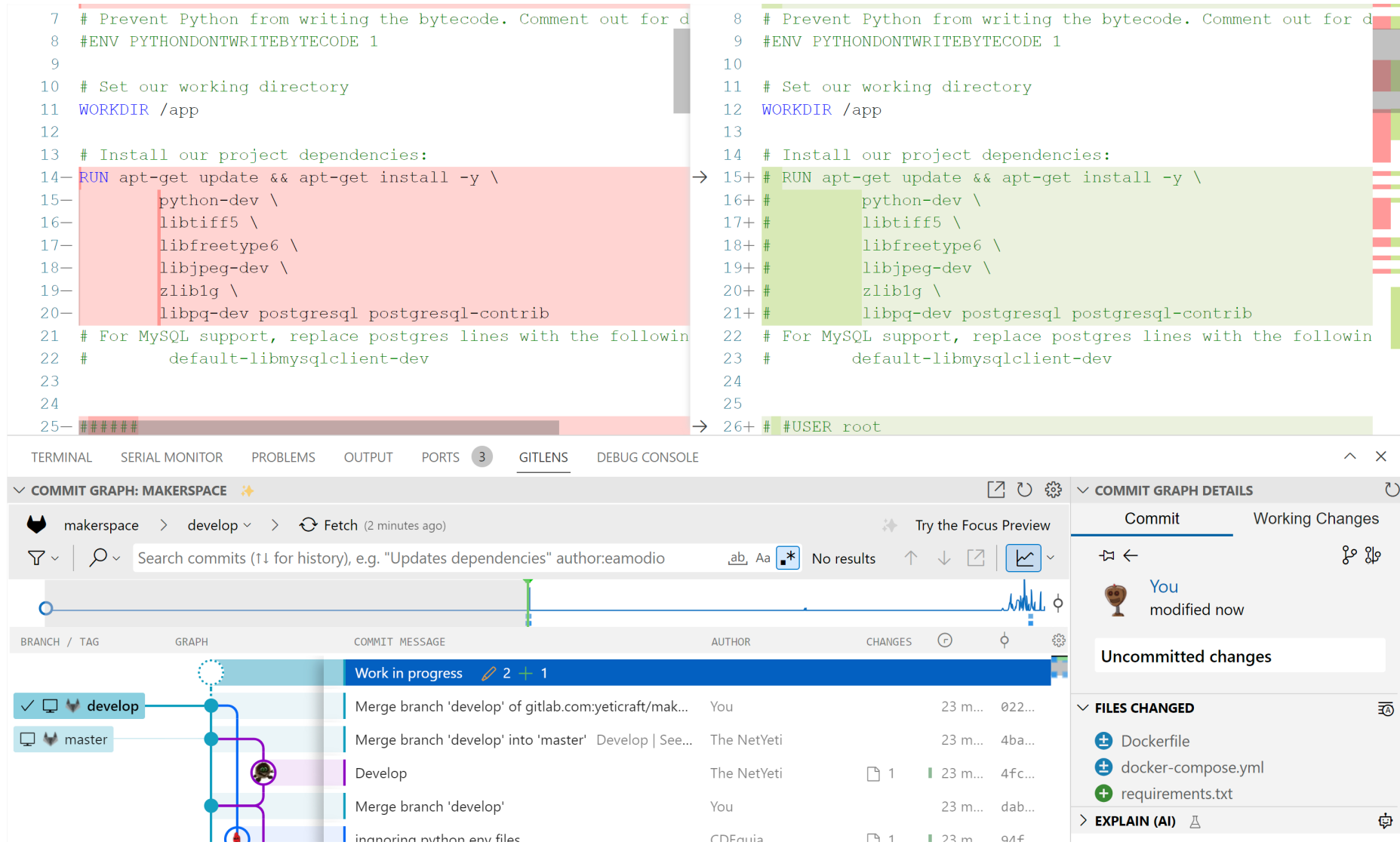


- `git add`
  - The `git add` command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit.
  - However, `git add` doesn't really affect the repository in any significant way—changes are not actually recorded until you run [`git commit`](#).
- `git commit`
  - The `git commit` command captures a snapshot of the project's currently staged changes.
  - Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to.

# Saving changes

- 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.
- git stash
  - git stash temporarily shelves (or *stashes*) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.
  - Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

# Git diff



```
7 # Prevent Python from writing the bytecode. Comment out for d
8 #ENV PYTHONDONTWRITEBYTECODE 1
9
10 # Set our working directory
11 WORKDIR /app
12
13 # Install our project dependencies:
14- RUN apt-get update && apt-get install -y \
15-     python-dev \
16-     libtiff5 \
17-     libfreetype6 \
18-     libjpeg-dev \
19-     zlib1g \
20-     libpq-dev postgresql postgresql-contrib
21 # For MySQL support, replace postgres lines with the followin
22 #     default-libmysqlclient-dev
23
24
25- #####
```

```
8 # Prevent Python from writing the bytecode. Comment out for d
9 #ENV PYTHONDONTWRITEBYTECODE 1
10
11 # Set our working directory
12 WORKDIR /app
13
14 # Install our project dependencies:
15+ # RUN apt-get update && apt-get install -y \
16+ #     python-dev \
17+ #     libtiff5 \
18+ #     libfreetype6 \
19+ #     libjpeg-dev \
20+ #     zlib1g \
21+ #     libpq-dev postgresql postgresql-contrib
22 # For MySQL support, replace postgres lines with the followin
23 #     default-libmysqlclient-dev
24
25
26+ #USER root
```

TERMINAL SERIAL MONITOR PROBLEMS OUTPUT PORTS 3 GITLENS DEBUG CONSOLE

▼ COMMIT GRAPH: MAKERSPACE

makerspace > develop > Fetch (2 minutes ago)

Search commits (↑ for history), e.g. "Updates dependencies" author:eamodio

ab, Aa No results

BRANCH / TAG GRAPH COMMIT MESSAGE AUTHOR CHANGES

✓ develop Merge branch 'develop' of gitlab.com:yeticraft/mak... You 23 m... 022...

master Merge branch 'develop' into 'master' Develop | See... The NetYeti 23 m... 4ba...

Develop Merge branch 'develop' The NetYeti 1 23 m... 4fc...

ignoring python env files You 23 m... dab...

CDEquia 1 23 m 94f

Commit Working Changes

You modified now

Uncommitted changes

▼ FILES CHANGED

- ± Dockerfile
- ± docker-compose.yml
- + requirements.txt

> EXPLAIN (AI)

# Inspecting a repo

- git status
  - The git status command displays the state of the working directory and the staging area.
  - It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.
  - Status output does *not* show you any information regarding the committed project history. For this, you need to use [git log](#).
- git blame
  - Examine specific points of a file's history and get context as to who the last author was that modified the line.
  - Explore the history of specific code and answer questions about what, how, and why the code was added to a repository.

# Undoing Commits & Changes

- Finding what is lost: Reviewing old commits
  - git log
    - The git log command displays committed snapshots. It lets you list the project history, filter it, and search for specific changes.
  - git checkout
    - The git checkout command lets you navigate between the branches created by git branch. Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch.



# Undoing Commits & Changes

- Undoing a committed snapshot
  - git revert
    - "Undo command". Instead of removing the commit from the project history, it figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content.
  - git commit --amend
  - git reset (next slide)

# Git reset

- A versatile git command undoing changes. The git reset command has a powerful set of options but we'll just be using the following reset modes for this tutorial:
  - --soft: Only resets the HEAD to the commit you select. Works basically the same as git checkout <commit #> but does not create a detached head state.
  - --mixed: Resets the HEAD to the commit you select in both the history and undoes the changes in the index.
  - --hard: Resets the HEAD to the commit you select in both the history, undoes the changes in the index, and undoes the changes in your working directory.

# Git Syncing

- git remote
  - The git remote command lets you create, view, and delete connections to other repositories. Remote connections are more like bookmarks rather than direct links into other repositories.
- git pull
  - The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.
- git push
  - The git push command is used to upload local repository content to a remote repository.
- git fetch
  - The git fetch command downloads commits, files, and refs from a remote repository into your local repo.

# Pull request

- In their simplest form, pull requests are a mechanism for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their hosted SCM account. This lets everybody involved know that they need to review the code and merge it into the main branch.
- But, the pull request is more than just a notification—it's a dedicated forum for discussing the proposed feature. If there are any problems with the changes, teammates can post feedback in the pull request and even tweak the feature by pushing follow-up commits. All of this activity is tracked directly inside of the pull request.

# Using Branches

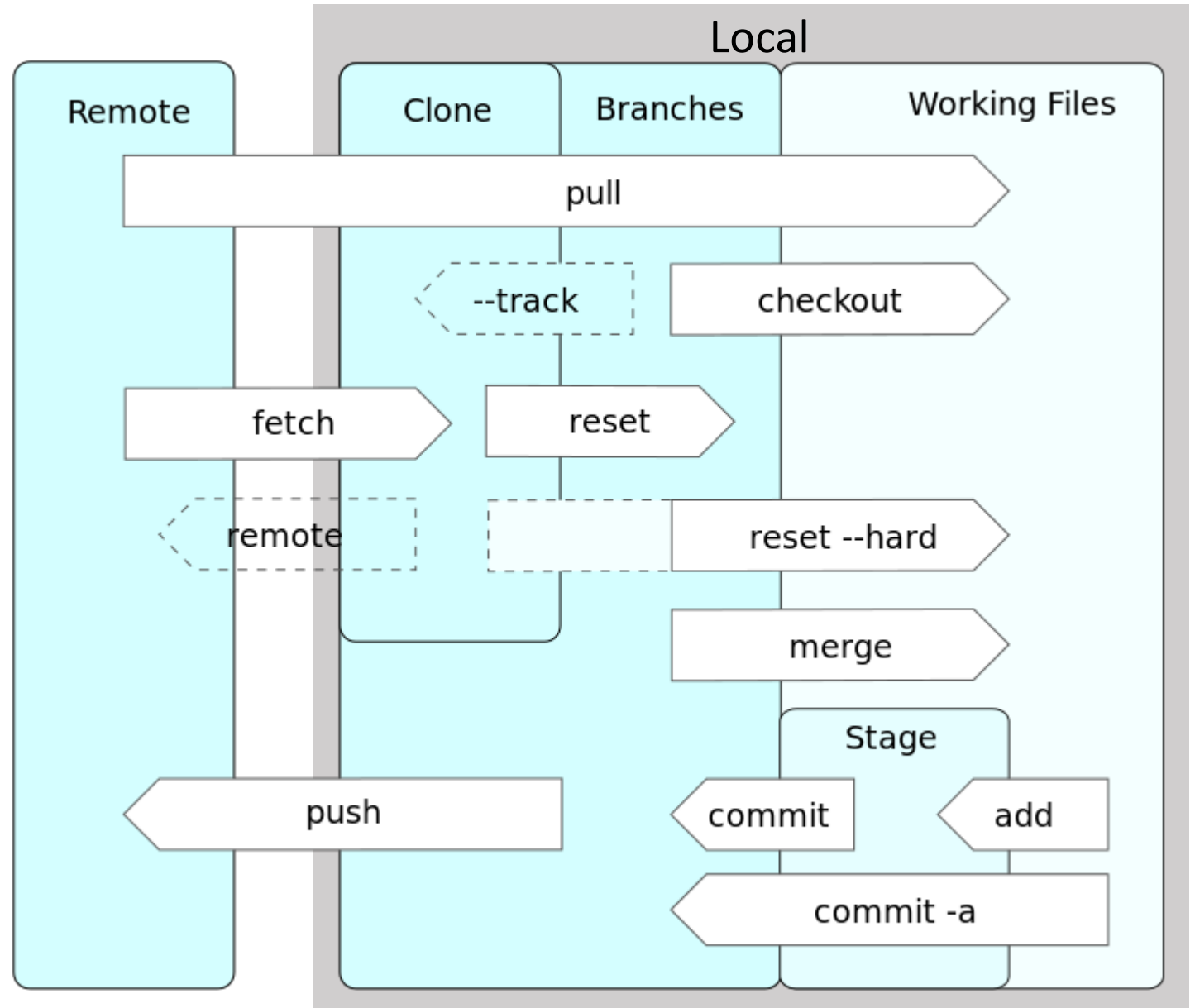
- git branch
- git checkout
- git merge

# Merging

- A “**merge**” is the act of incorporating the changes from one branch or commit into a second branch.
- The histories do not have to match exactly; Git will work to sort them out.
- If a merge fails, Git will notify you of a “**merge conflict**”.
- Merge conflicts must be fixed manually, and then **added** and **committed**.
- “**git status**” will show any merge conflicts.
- A successful merge creates a new commit automatically.
- What causes a merge conflict?
  - 2 branches modify the same place in the same file.
  - This often happens at the end of files or between function declarations.
  - Git does not know which version of the change you want, or if you want both of them consecutively (and in which order), so you must inspect the conflict and resolve it.
- If you cannot **push** your code to a **remote**, you probably need to first **pull** the latest changes, resolve any existing **merge conflicts**, then try to **push** again.
- **Small** commits are better than large ones!

# Git Workflow

- You **pull** from a remote to your local repository.
  - A **pull** is a **fetch** and a **merge**
  - Option to use fetch and merge separately and avoid pull.
- You **push** a branch from your local repository to a remote.
- A failure to merge is a **merge conflict**, and must be resolved manually.

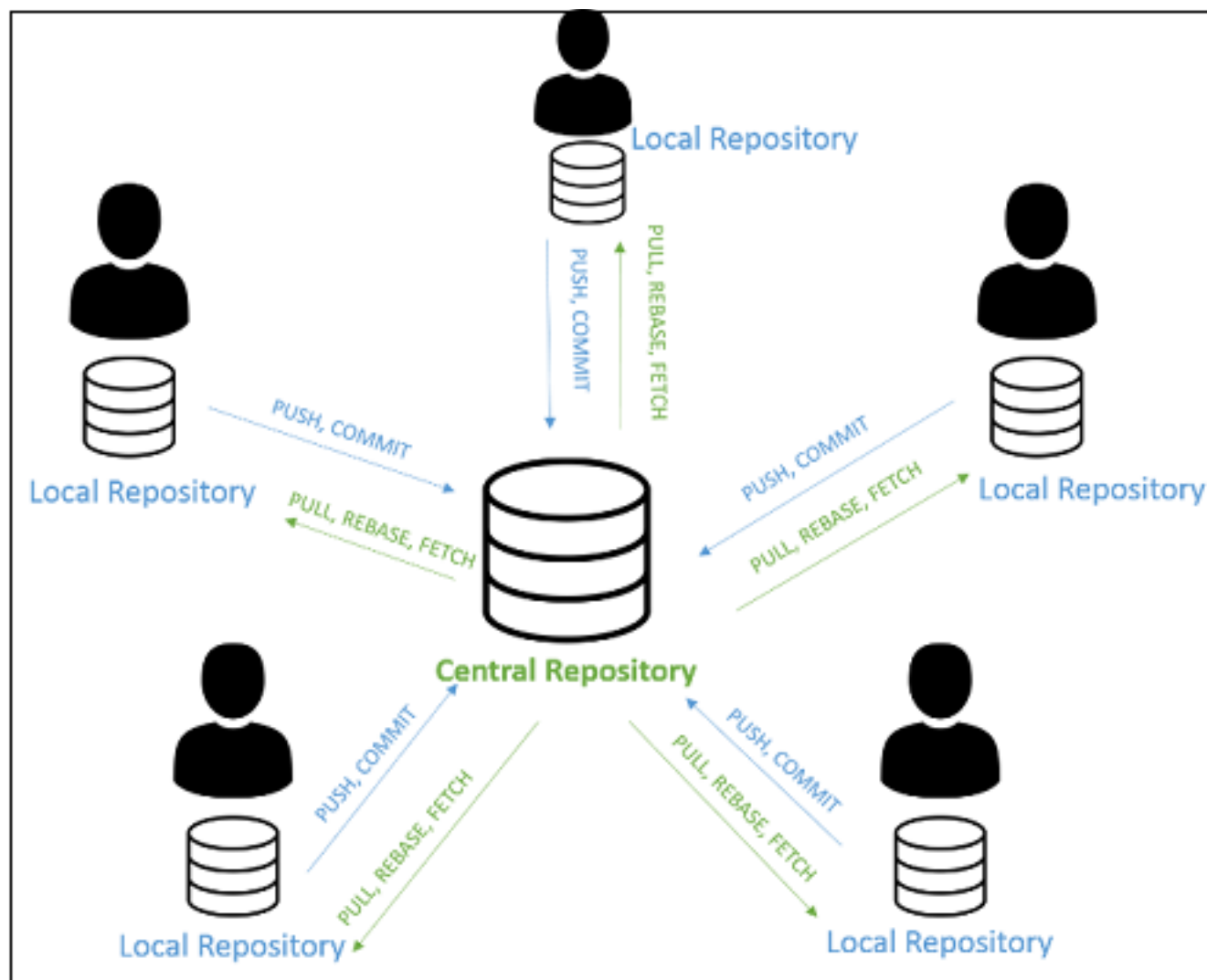


# Comparing Git Workflows

- Centralized workflow
  - There is only one central main repository. Each developer clones the repository works locally on the code makes a commit with changes, and pushes it to the central main repository for other developers to pull and use in their work.



# Centralized Workflow

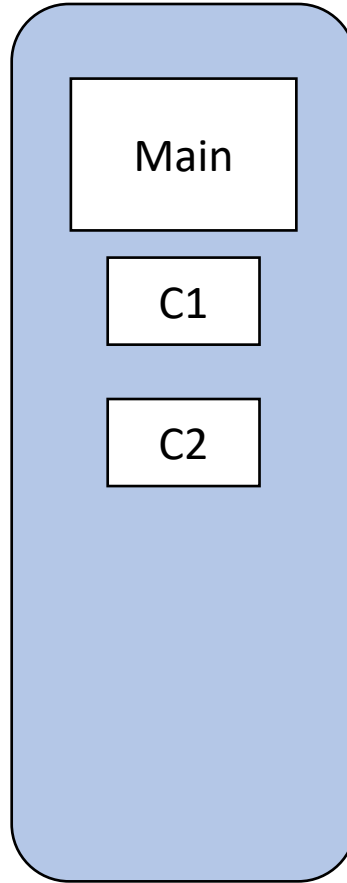
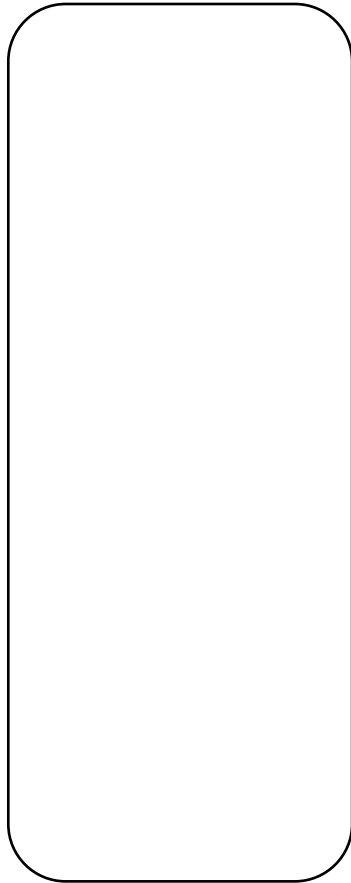


# Collaboration with GitHub

Remote Repo

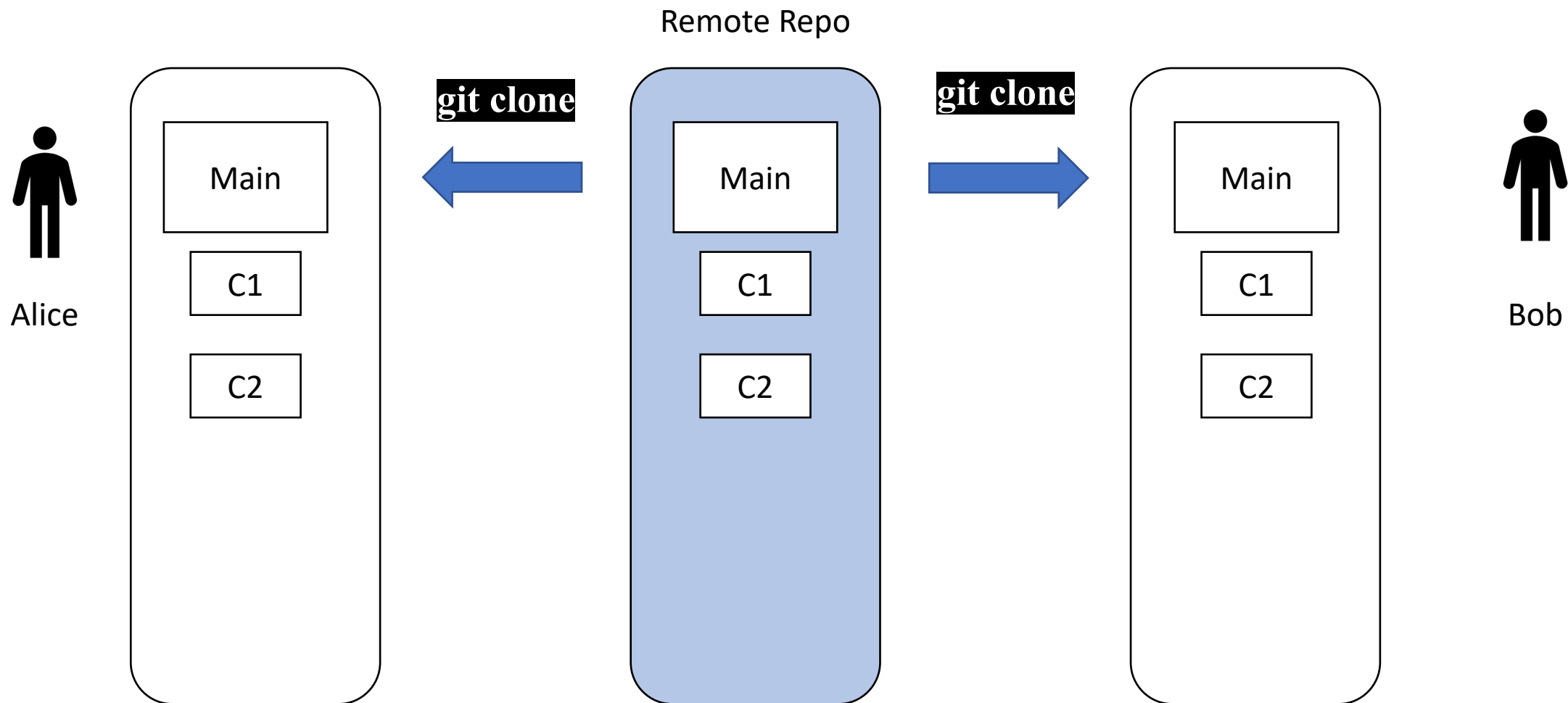


Alice



Bob

# Collaborate

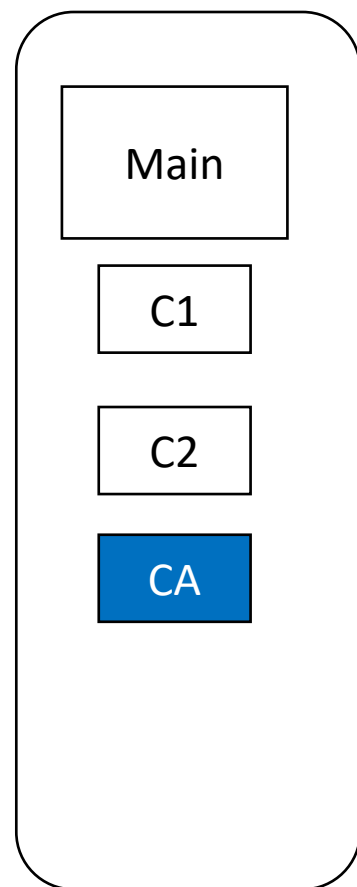


# Collaborate

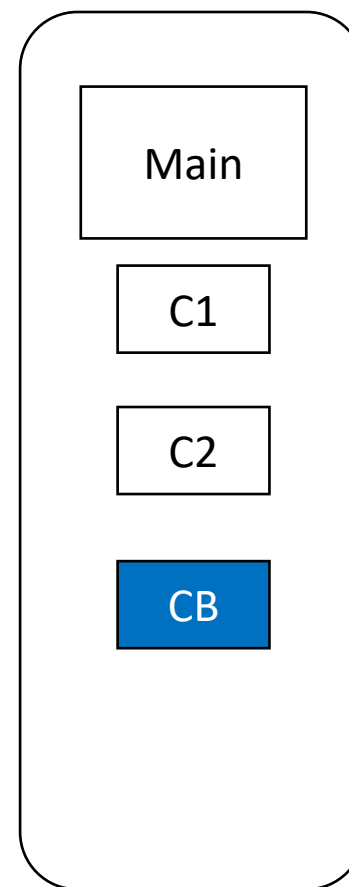
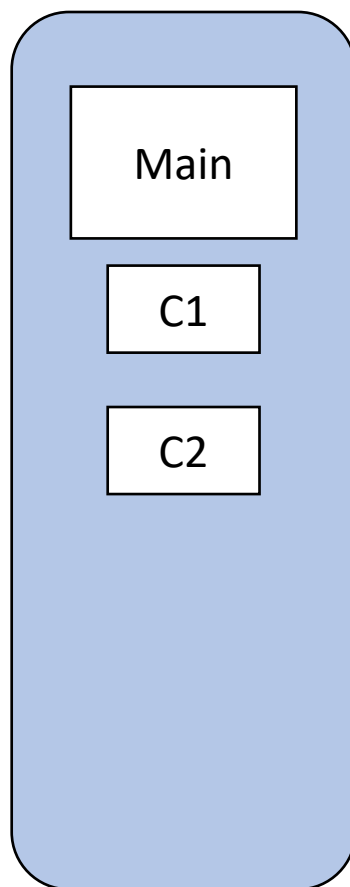
Remote Repo



Alice



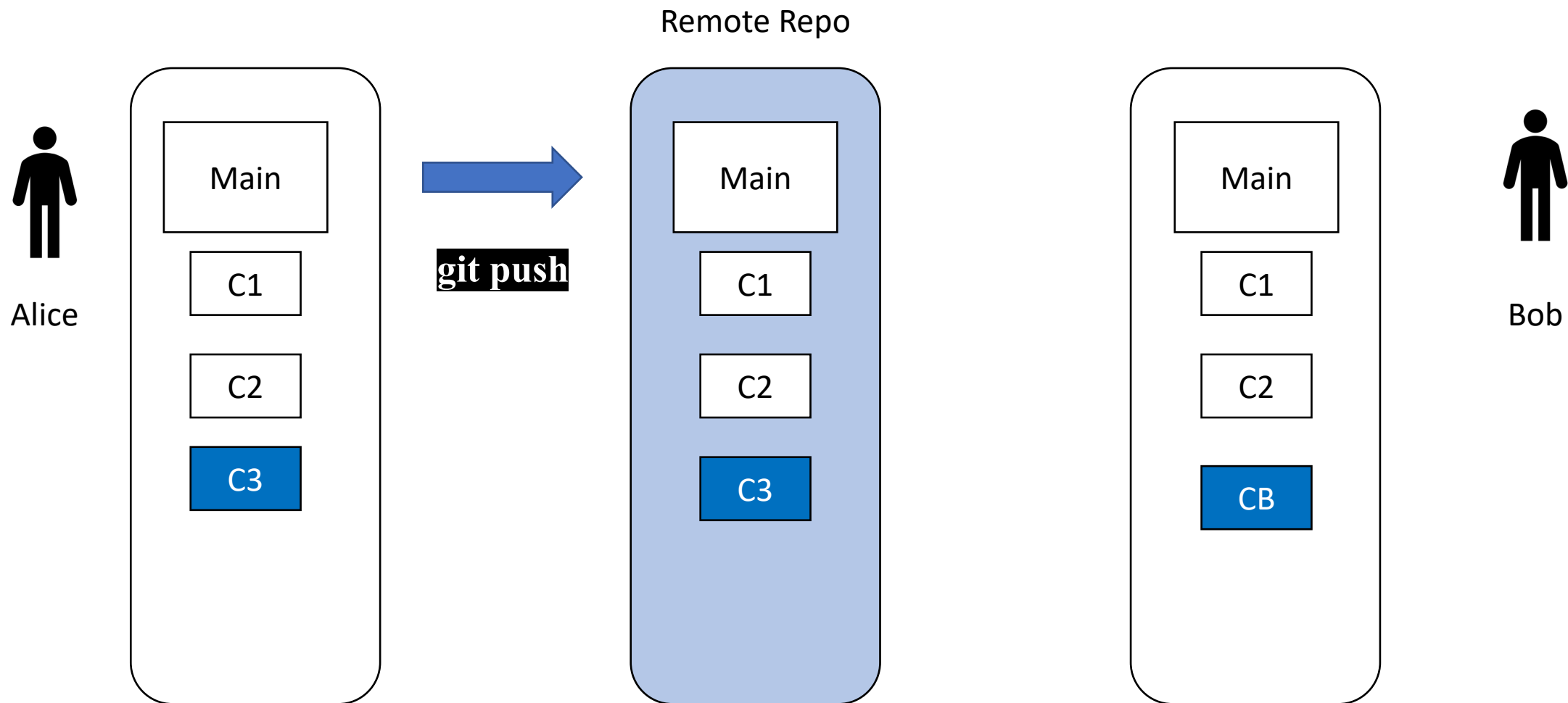
**git add**  
**git commit**



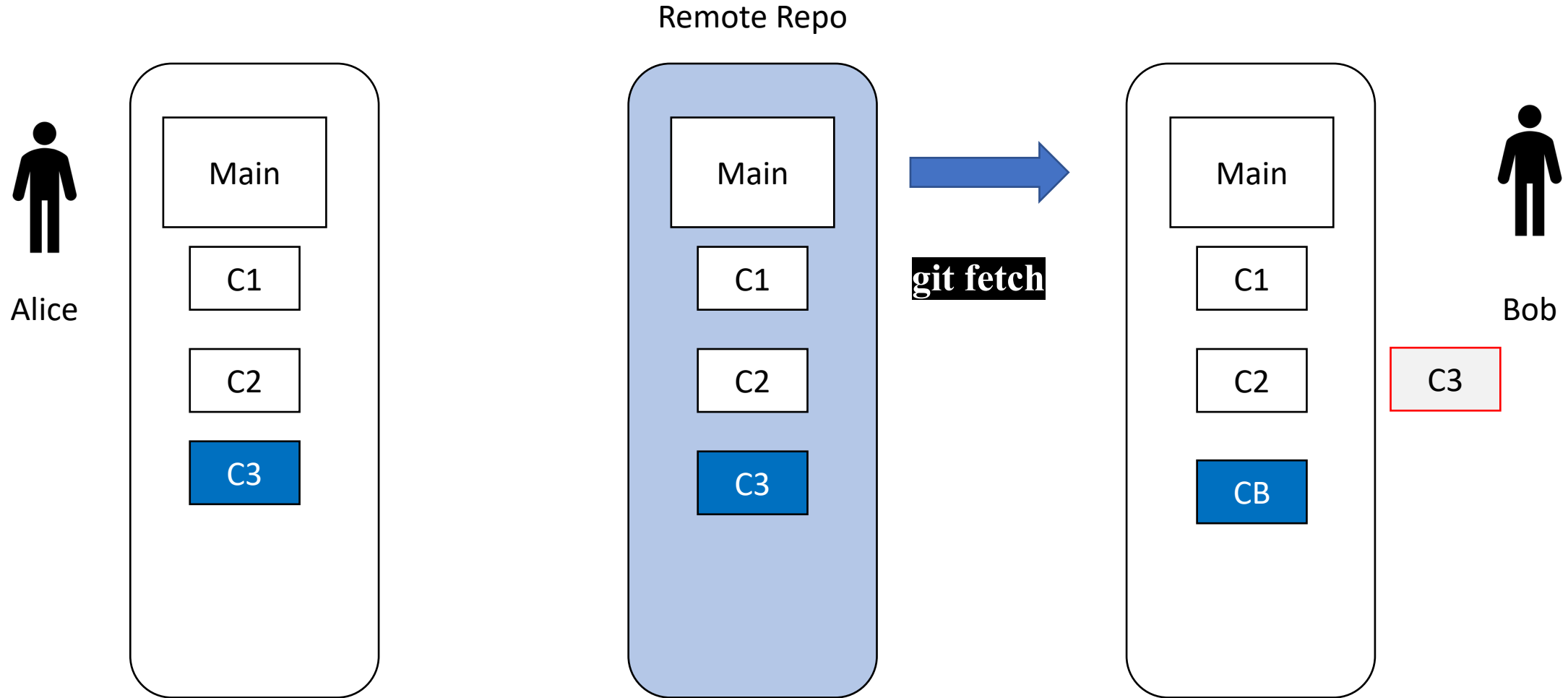
Bob

**git add**  
**git commit**

# Collaborate



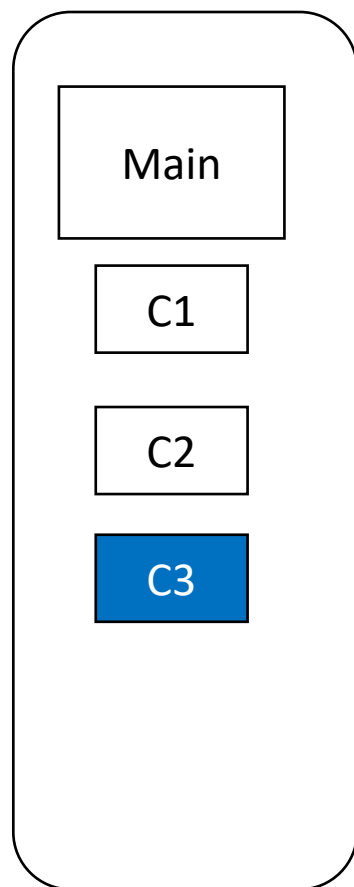
# Collaborate



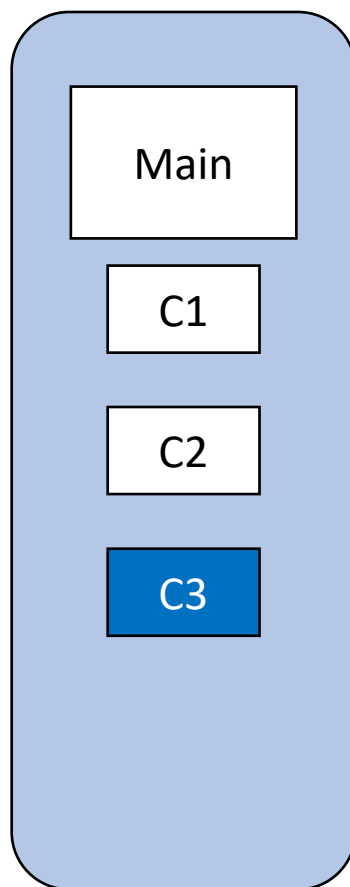
# Collaborate



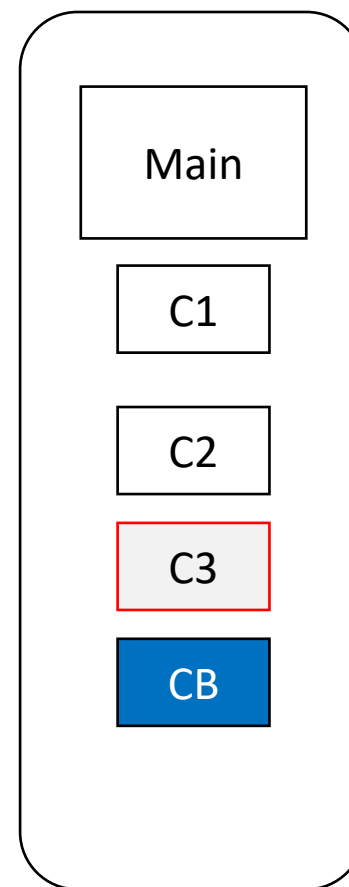
Alice



Remote Repo

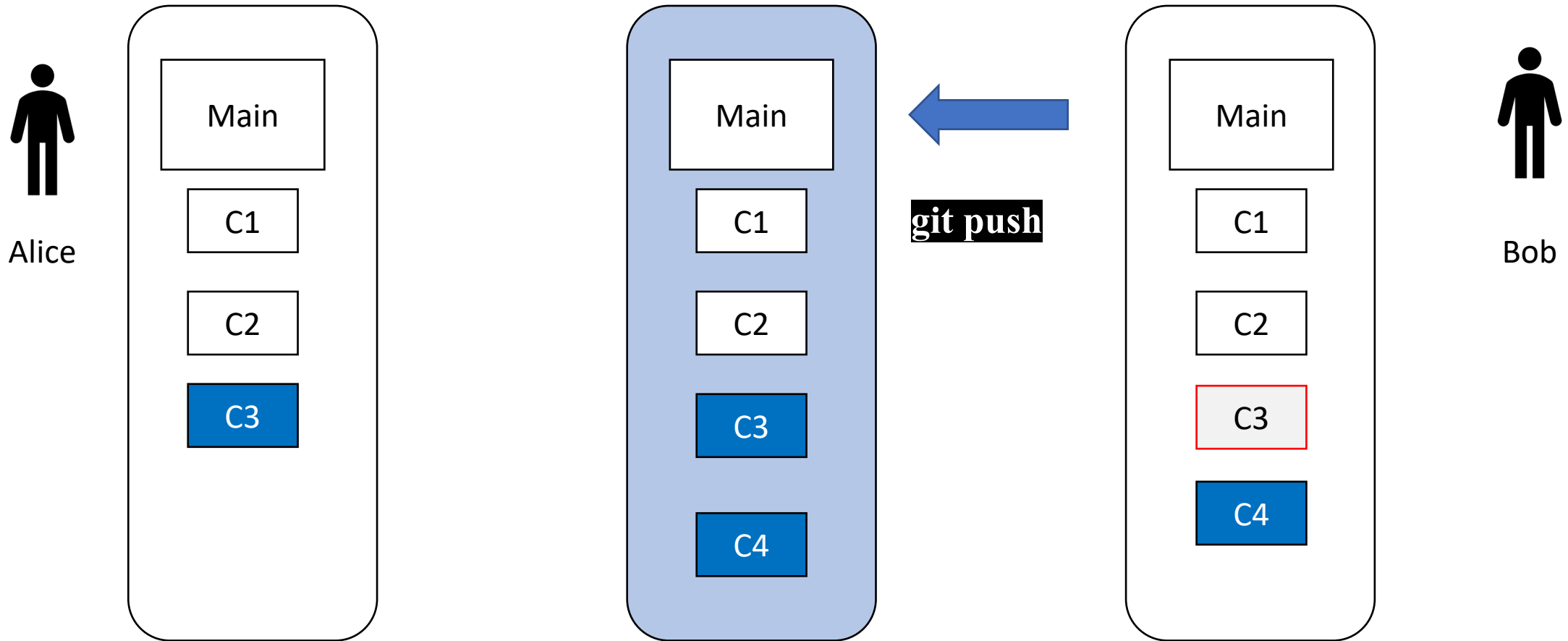


**git merge**



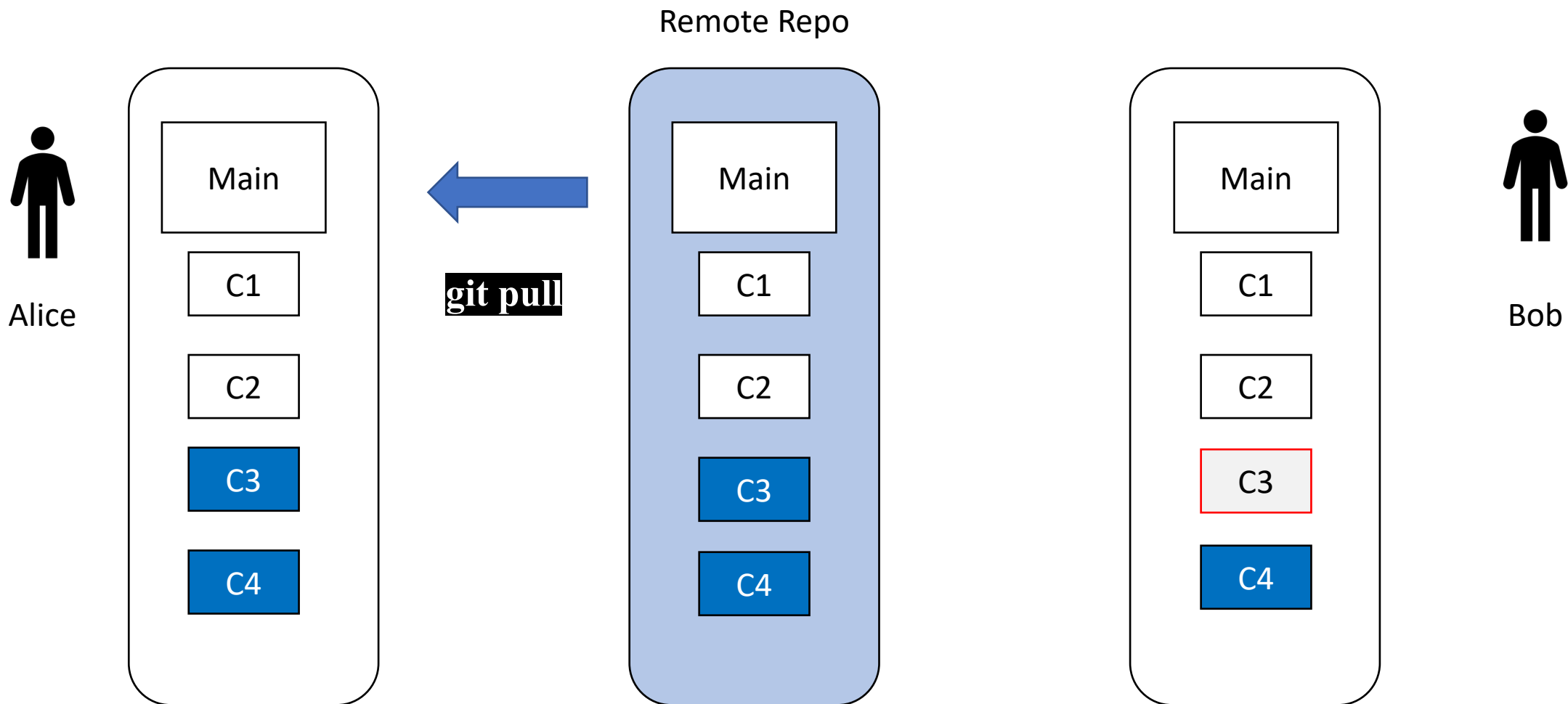
Bob

# Collaborate





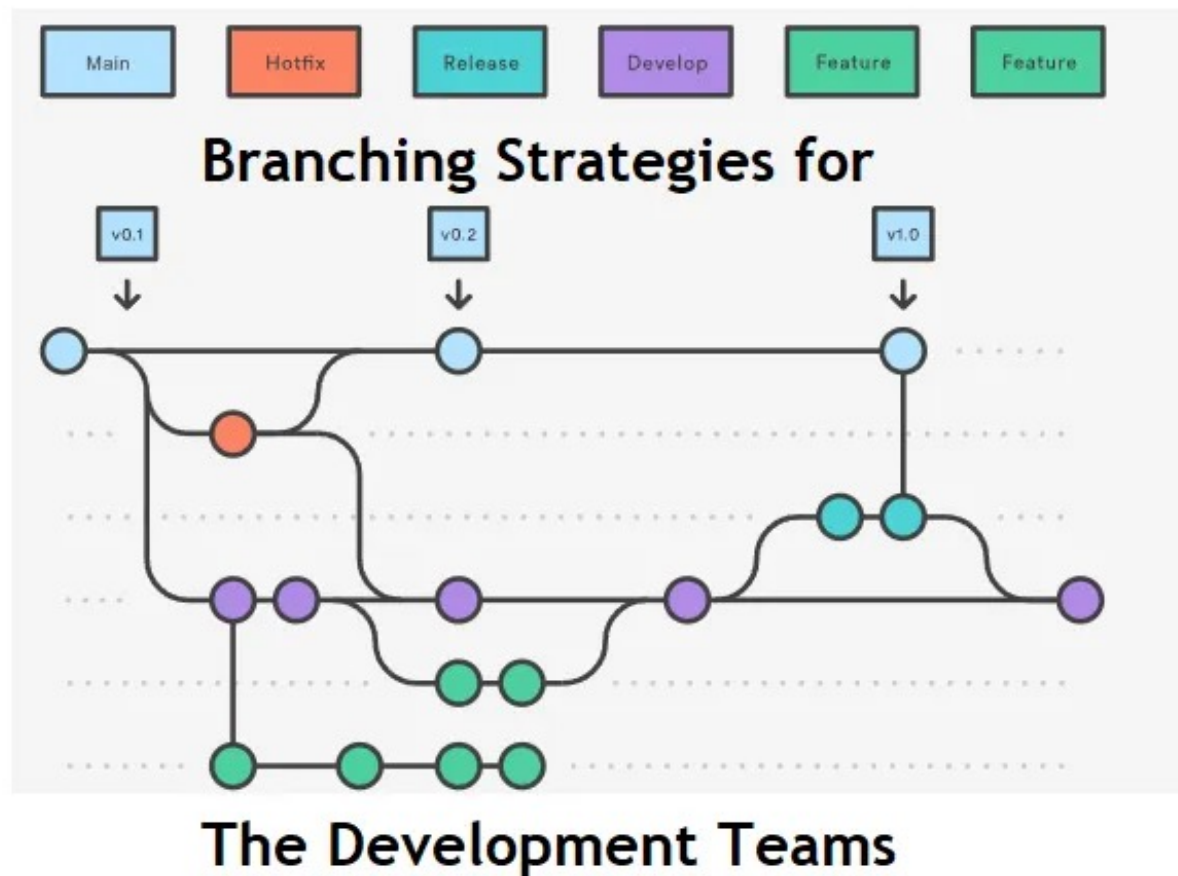
# Collaborate



# Comparing Git Workflows

- Feature branching
  - Feature Branching is a logical extension of Centralized Workflow. The core idea behind the [Feature Branch Workflow](#) is that all feature development should take place in a dedicated branch instead of the master branch. This encapsulation makes it easy for multiple developers to work on a particular feature without disturbing the main codebase. It also means the master branch should never contain broken code, which is a huge advantage for continuous integration and deployment (CI/CD) environments.

# Feature Branch Workflow

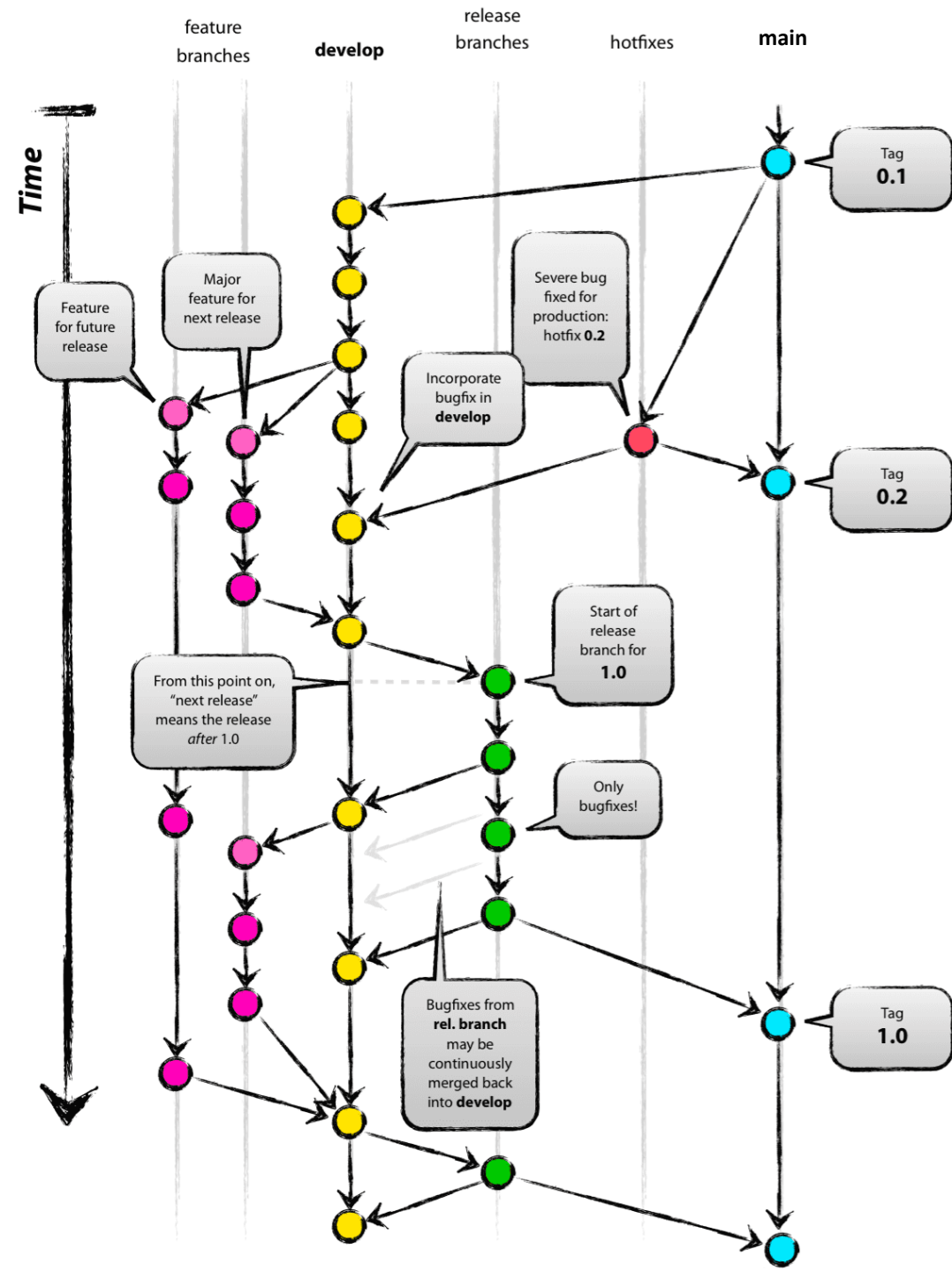


# Comparing Git Workflows

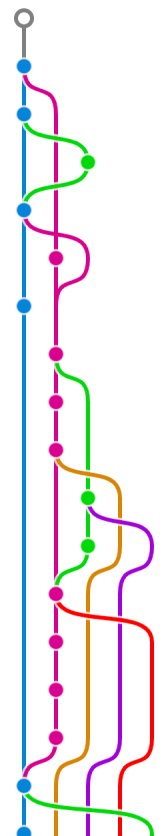
- Gitflow Workflow
  - The [Gitflow Workflow](#) was first published in a highly regarded 2010 blog post from [Vincent Driessen at nvie](#). The Gitflow Workflow defines a strict branching model designed around the project release. This workflow doesn't add any new concepts or commands beyond what's required for the Feature Branch Workflow. Instead, it assigns very specific roles to different branches and defines how and when they should interact.



# Gitflow Workflow



# Git Workflow

Graph	Description	Date	Author	Commit
	<b>Uncommitted Changes (3)</b>	21 Jan 2024 12:38	*	*
	<b>develop</b> <b>origin</b> Merge branch 'develop' of gitlab.com:yeticraft/makerspace in...	27 Feb 2022 11:44	Garth Johnson	022be108
	<b>master</b> <b>origin</b> <b>origin/HEAD</b> Merge branch 'develop' into 'master'	27 Feb 2022 11:35	The NetYeti	4ba9126c
	Develop	27 Feb 2022 11:35	The NetYeti	4fcb83cb
	Merge branch 'develop'	27 Feb 2022 11:34	Garth Johnson	dabac4a3
	ingnoring python env files	26 Feb 2022 11:48	CDEguia	94f6cd9d
	Merge branch 'adminviews'	26 Feb 2022 12:54	Garth Johnson	f6b18213
	Merge branch 'adminviews' into develop	26 Feb 2022 12:49	Garth Johnson	a16643a0
	Merge branch 'adminviews'	26 Feb 2022 12:48	Garth Johnson	b4a4f535
	Merge branch 'newsite' into develop	26 Feb 2022 12:45	Garth Johnson	17c0ce0e
	<b>adminviews</b> <b>origin</b> Merge branch 'adminviews' of gitlab.com:yeticraft/makerspa...	26 Feb 2022 12:11	Garth Johnson	d6ae5185
	Add the Profile model to the admin page	11 Mar 2020 12:09	CDEguia	5a6c86b1
	Merge branch 'develop' of gitlab.com:yeticraft/makerspace into develop	26 Feb 2022 11:54	Garth Johnson	3881aa4d
	fix conditionals inside initializer	6 Mar 2020 13:54	CDEguia	3fcd0d50
	Intake cli definitions	5 Mar 2020 16:43	CDEguia	a45e60e1
	spelling fixes	5 Mar 2020 15:37	CDEguia	7c2b11c2
	<b>origin/1.0</b> <b>v1.0.0</b> Merge branch '1.0' into 'master'	27 Jan 2021 09:51	The NetYeti	41b63912
	Installing GitPod	25 Jan 2021 16:04	The NetYeti	9219efb5

TERMINAL
SERIAL MONITOR
PROBLEMS
OUTPUT
PORTS 2
DEBUG CONSOLE

zsh - makerspace
+
⌵
📄
🗑️
⋮
⬆️
✕

```

❏ / ❏ ~/Projects/BMS/makerspace / on ❏ ❏ develop !2 ?1
❏ / ❏ ~/Projects/BMS/makerspace / on ❏ ❏ develop !2 ?1
>

```

at 12:39:01

# Guidelines

- Short-lived branches
  - The longer a branch lives separate from the production branch, the higher the risk for merge conflicts and deployment challenges. Short-lived branches promote cleaner merges and deploys.
- Minimize and simplify reverts
  - It's important to have a workflow that helps proactively prevent merges that will have to be reverted. A workflow that tests a branch before allowing it to be merged into the main branch is an example. However, accidents do happen. That being said, it's beneficial to have a workflow that allows for easy reverts that will not disrupt the flow for other team members.
- Match a release schedule / cadence
  - A workflow should complement your business's software development release cycle. If you plan to release multiple times a day, you will want to keep your main branch stable. If your release schedule is less frequent, you may want to consider using Git tags to tag a branch to a version.

# Common Industry Programmer Workflow

1. Choose a bug/feature to work on
  2. **Checkout** the appropriate branch
  3. **Pull** the latest additions from the main repository
  4. Create a **branch** for the bug/feature request
  5. Do the coding necessary & **commit**
  6. **Push** the changes to your remote
  7. Create a **pull request** to main repo
- The **pull request** is where other programmers review your code & make comments/suggestions.
  - If changes are needed to your code, then repeat the process.
- **Never commit directly to the main branch.**



# Practical Guidelines

## • Do's

- Make small, incremental commits (within reason) are good. Avoid monolithic commits at all cost.
- Use a separate branch for each new bug/feature request.
- Write nice commit messages. Otherwise, the commit log is useless.
- Use a [.gitignore](#) to keep cruft out of your repo.
- Search engines are your friend. Someone else has had the same question/mistake/situation as you, and they have already asked the question online. It's probably on StackOverflow.

## • Don'ts

- Do not commit commented-out debug code.
  - It's messy. It's ugly. It's unprofessional.
- Do not mix your commits. (e.g., Don't commit two bugfixes at the same time.)
- Do not commit sensitive information (passwords, database dumps, etc.)
- Do not commit whitespace differences, unless it is specifically needed.
- Do not commit large binaries.

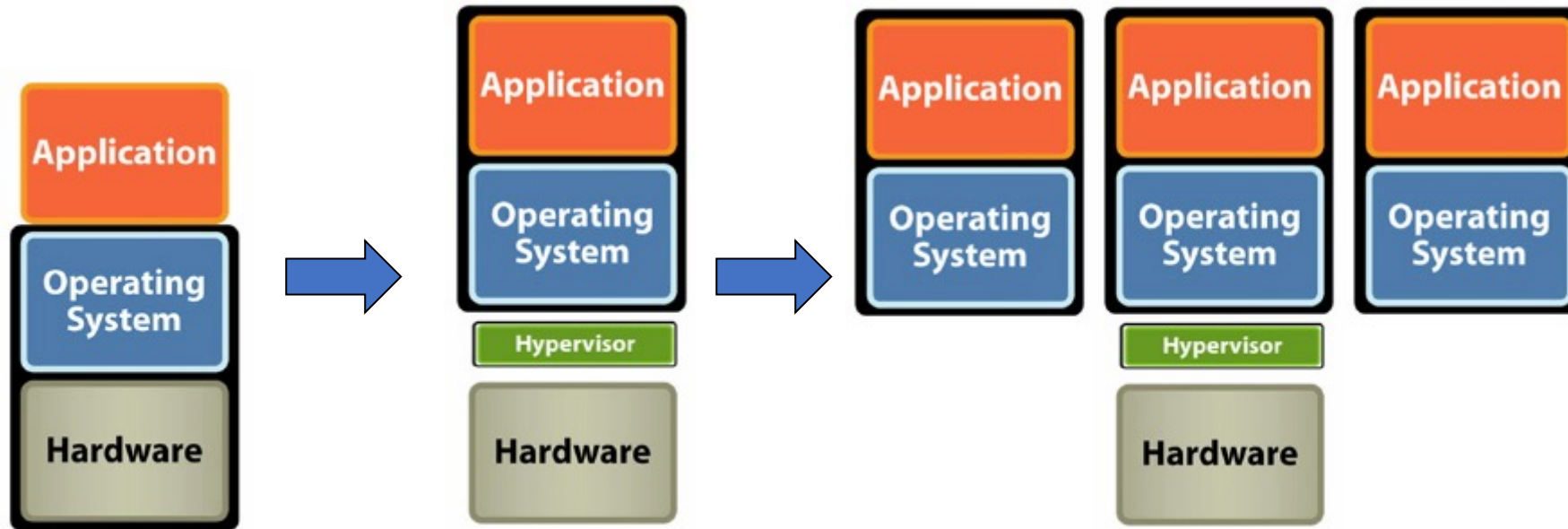
# What is Virtualization?

- An abstraction
- Usually performed via software
- Many different types
  - Hardware
  - Software
  - Data
  - Network
- Our focus will be hardware virtualization

# Hardware Virtualization

- Abstracts underlying physical hardware from operating systems and applications
- Allows multiple guest operating systems to run in parallel
- Physical resources are shared amongst all guest OS and virtualization software

# Virtualization



# Terminology

- Host Machine
  - The physical hardware/server
- Hypervisor
  - The virtualization software
  - Acts as the true 'OS' for the server
- Virtual Machines
  - Instances of the virtualized OS
  - Sometimes called Guest OS

# Why Virtualize?

- Low CPU and memory utilization
- Overpowered and overpriced hardware
- Datacenter sprawl
- Power and HVAC
- High administrative labor costs
- OS Licensing

# Benefits

- Fewer servers, with better system utilization
- Easier redundancy and disaster recovery
- Decreased downtime
- Ability to leverage patch management
- Greatly decreased build times
- Excellent testbed
- Sharing of pooled resources

# Virtualization Software

- Runs operating systems in fully emulated environment
  - Vmware (Vmware Inc.)
  - **VirtualBox (Oracle)**
  - Hyper-V (Microsoft)
  - Xen (open source project)



# Virtualization Terminology

- ***Host OS*** – running on physical computer
  - Only one host OS may run at a time
  - “Hosts” the other running operating systems
- ***Guest OS*** – running in emulated environment
  - Can run multiple guests at the same time
  - Guest ***thinks*** it is running on actual hardware
- ***Virtual machine*** – set of files that make up a guest OS

# Virtual Machine Advantages

- Can distribute a pre-configured OS
  - Run VM, install/configure it, then export to another VM image
- Easy to create multiple snapshots
  - If something goes wrong, roll-back to a previously saved snapshot
- Portable
  - Run on any host OS
  - Store on portable hard drive or laptop
- Allows running of automated system services
  - Unlike containers (but similar to host OS)

# Virtual Machine Advantages

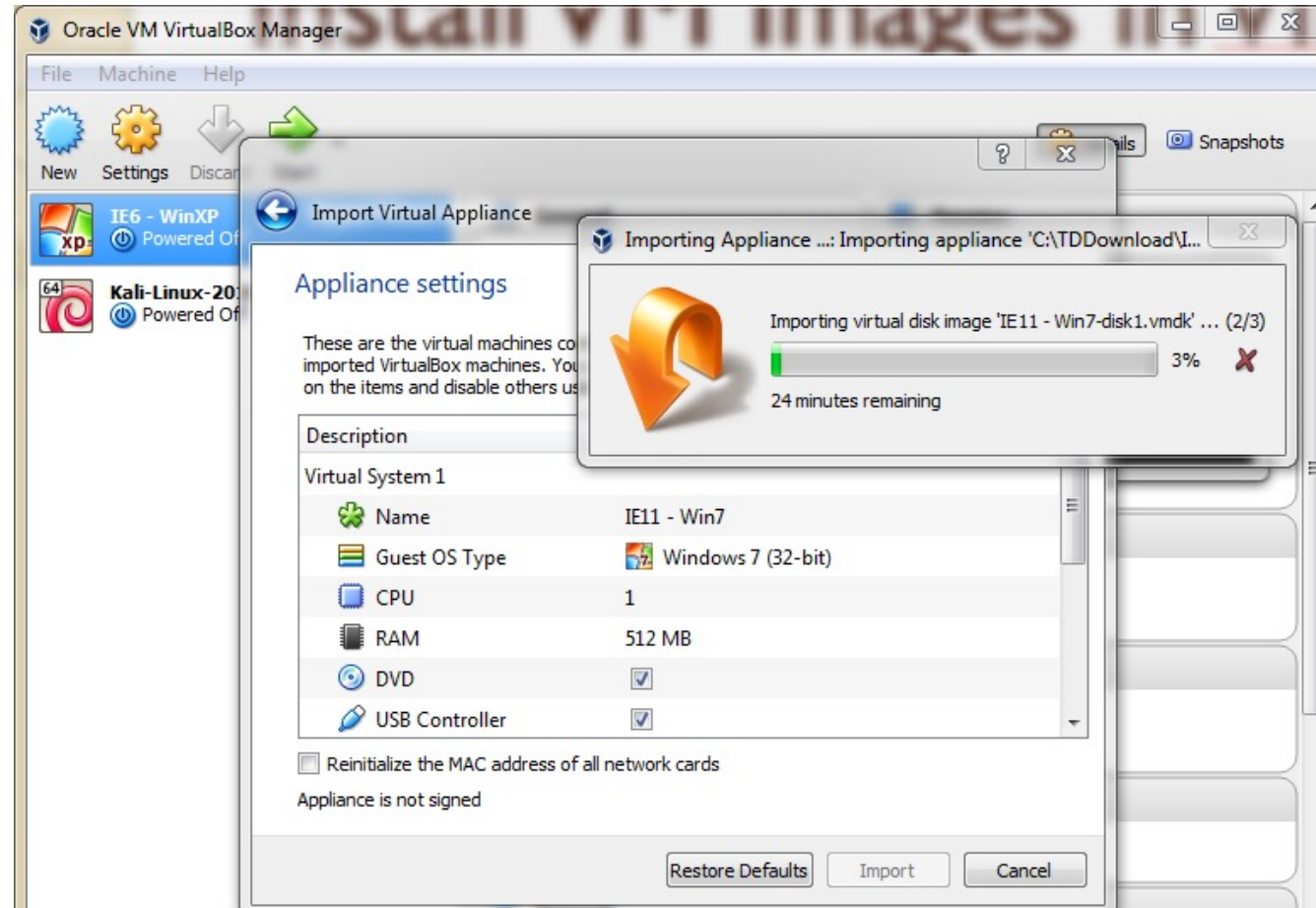
- Sandbox
  - Does not affect anything on host OS
- Networked
  - Can access over the network

# Install VM Images in VirtualBox

- For VM images with .ova file type
  - VirtualBox menu:
    - “File” → “Import Appliance”
    - Choose the \*.ova image file to import the VM image
    - Just use the default configurations



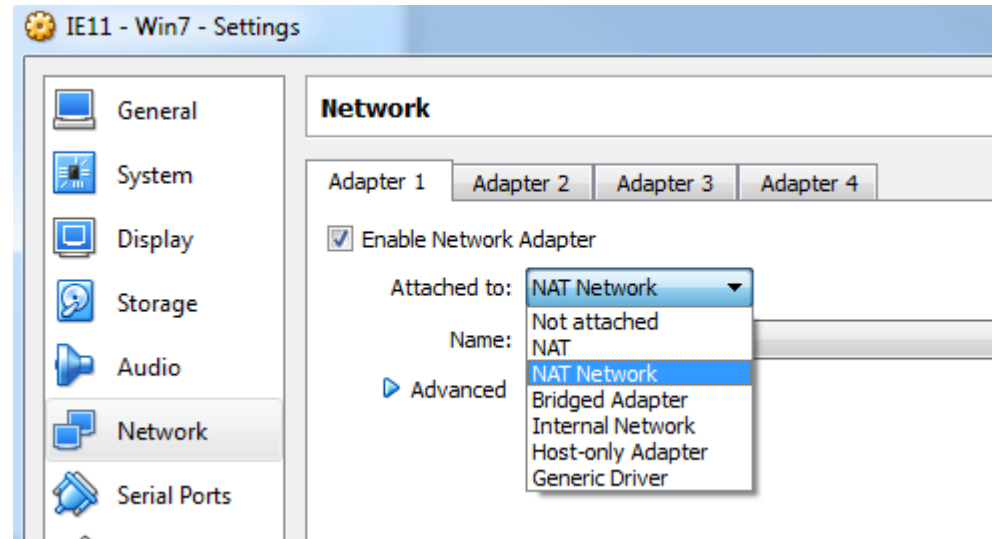
# Importing Win7 VM Image....



- Take a while, so be patient.... ( a few minutes)

# Networking in VirtualBox

- VirtualBox provides the following networking options:



- We will introduce:
  - NAT, NAT Network, Bridged Adapter



# IP Address Checking Tool

In Windows, run “ipconfig” under “cmd” window

```
C:\Windows\system32\cmd.exe
C:\Users\Student Admin>
C:\Users\Student Admin>ipconfig

Windows IP Configuration

Wireless LAN adapter Wireless Network Connection:

    Connection-specific DNS Suffix  . : user.wls.ucf.edu
    Link-local IPv6 Address . . . . . : fe80::e080:5653:235c:8c61%15
    IPv4 Address. . . . . : 10.32.218.81
    Subnet Mask . . . . . : 255.255.192.0
    Default Gateway . . . . . : 10.32.192.1

Ethernet adapter Bluetooth Network Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter Local Area Connection:
```

In Linux, run “ifconfig” in terminal (old) or "ip a" (new)

```
root@kali: ~
File Edit View Search Terminal Help

root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fefa:258e prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:fa:25:8e txqueuelen 1000 (Ethernet)
    RX packets 6 bytes 1580 (1.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 23 bytes 2189 (2.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
```

```
Windows PowerShell x /tmp
>

> ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default ql
    link/ether 00:15:5d:92:99:86 brd ff:ff:ff:ff:ff:ff
    inet 172.17.19.69/20 brd 172.17.31.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe92:9986/64 scope link
        valid_lft forever preferred_lft forever

>
```

# Networking Diagnosis Tool

- Use “Ping” command to check if a host is reachable
  - In Windows, run “ping x.x.x.x” under “cmd” window
  - In Linux, run “ping x.x.x.x” in terminal
    - Use CTRL+C to stop the pinging action

```
root@kali:~# ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=255 time=0.376 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=255 time=0.138 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=255 time=0.256 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=255 time=0.234 ms
^C
--- 10.0.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.138/0.251/0.376/0.084 ms
root@kali:~#
```



# Networking Diagnosis Tool

- Use “traceroute” (\*nix) or "tracert" (Windows) command to see where the connection dropped

```
Terminal — -tcsh — 80x24
[[Anns-MacBook-Pro:~] work% traceroute ucalgary.ca
traceroute to ucalgary.ca (136.159.96.125), 64 hops max, 52 byte packets
 1  router.lan (192.168.88.1)  1.505 ms  0.523 ms  0.427 ms
 2  docsis-gateway.cg.shawcable.net (10.0.0.1)  3.828 ms  1.192 ms  1.439 ms
 3  96.51.128.1 (96.51.128.1)  7.925 ms  8.897 ms  10.847 ms
 4  rc3no-be109-1.cg.shawcable.net (64.59.134.101)  11.695 ms  10.690 ms  11.410 ms
 5  24.244.57.241 (24.244.57.241)  9.490 ms  10.554 ms  7.353 ms
 6  24.83.252.73 (24.83.252.73)  5.000 ms  11.118 ms *
 7  * 24.83.251.209 (24.83.251.209)  17.512 ms
 8  rc3no-be214.cg.shawcable.net (24.244.57.1)  12.368 ms
 9  * * *
10  * fw1uofc.cg.bigpipeinc.com (206.174.203.106)  13.452 ms *
11  h66-244-233-20.bigpipeinc.com (66.244.233.20)  11.527 ms  9.512 ms
12  fw1uofc.cg.bigpipeinc.com (206.174.203.106)  4.748 ms
13  h66-244-233-20.bigpipeinc.com (66.244.233.20)  11.944 ms
14  h77.gpvpn.ucalgary.ca (136.159.199.77)  9.835 ms  9.925 ms
15  h77.gpvpn.ucalgary.ca (136.159.199.77)  11.235 ms  16.360 ms
    10.59.226.25 (10.59.226.25)  13.461 ms
    * 10.59.226.25 (10.59.226.25)  10.866 ms  14.261 ms
    10.17.184.1 (10.17.184.1)  11.009 ms  13.211 ms^C
[Anns-MacBook-Pro:~] work%
```

```
Windows PowerShell
PS C:\Users\garth> tracert google.com

Tracing route to google.com [142.250.69.206]
over a maximum of 30 hops:

  0  1 ms  1 ms  1 ms  router.lan [192.168.42.1]
  1  3 ms  1 ms  1 ms  50.35.236.73
  2  2 ms  3 ms  3 ms  lr1-fndlwaxa-b-be-14.bb.as20055.net [137.83.80.182]
  3  4 ms  4 ms  4 ms  lr1-fndlwaxa-a-be-15.bb.as20055.net [204.11.65.158]
  4  4 ms  8 ms  4 ms  lr1-mtvrwaxx-a-be-19.bb.as20055.net [204.11.66.42]
  5  5 ms  4 ms  4 ms  lr1-tgrdorc-a-be-13.bb.as20055.net [198.179.53.146]
  6  5 ms  4 ms  5 ms  lr1-myviwaxx-a-be-18.bb.as20055.net [204.11.66.39]
  7  4 ms  4 ms  9 ms  lr1-myviwaxx-b-be-11.bb.as20055.net [204.11.65.80]
  8  4 ms  4 ms  4 ms  lr1-evrtwaxf-a-be-14.bb.as20055.net [198.179.54.240]
  9  4 ms  4 ms  4 ms  cr2-evrtwaxc-a-be-14.bb.as20055.net [137.83.80.65]
10  * * * Request timed out.
11  4 ms  4 ms  4 ms  google-stllwawb-c.pni.as20055.net [107.191.239.11]
12  5 ms  4 ms  4 ms  142.251.70.103
13  4 ms  3 ms  4 ms  142.251.48.213
14  4 ms  4 ms  3 ms  sea30s08-in-f14.1e100.net [142.250.69.206]

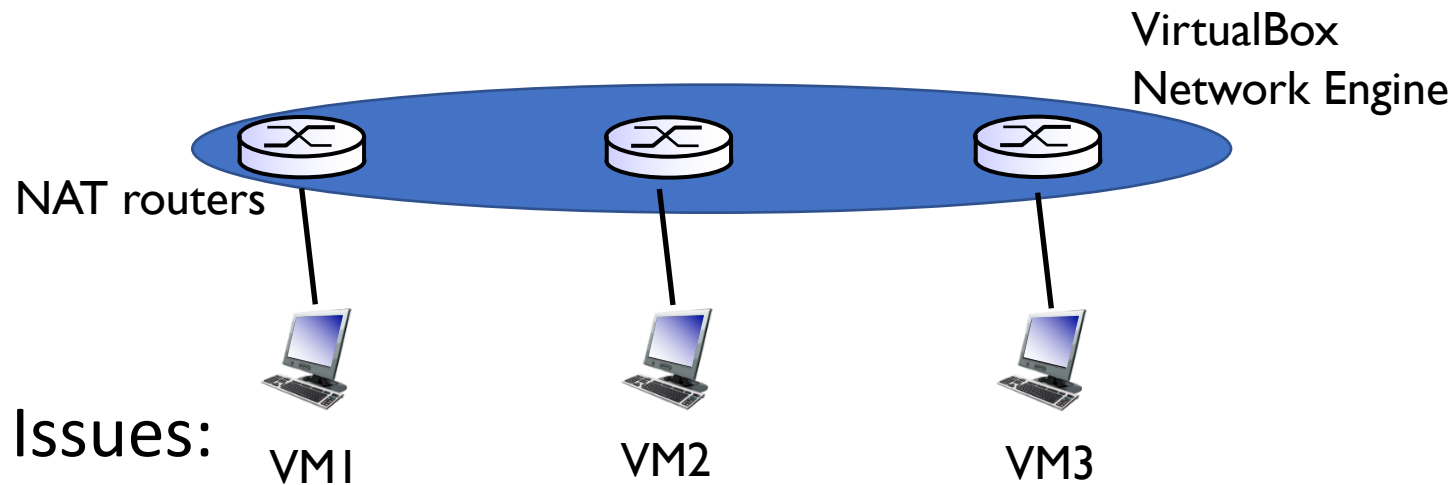
Trace complete.
PS C:\Users\garth>
```

# VirtualBox Networking Setup

- Objective:
  - Let multiple VMs in the same LAN
    - This LAN is private, cannot be connected from outside (for security purpose)
  - Each VM has Internet access
    - So that we can download/install software on them
- Two types of networks:
  - (Bridged Adapter) Host machine and VMs are in the same LAN
  - (NAT Network) Guest VMs in the LAN, cannot see host OS

# Networking in VirtualBox: NAT

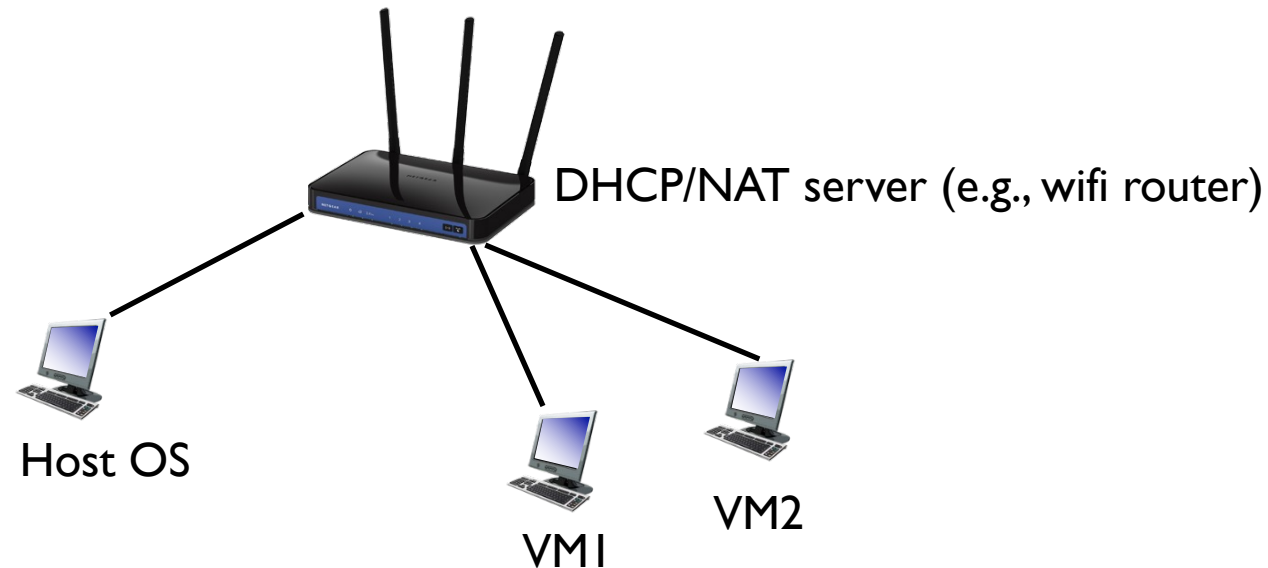
- Default configuration
- Virtualbox generates NAT routers
  - One NAT router for each VM
- Simplest, no configuration at all



- Issues:
  - Each VM in its own private LAN, cannot see each other

# Networking in VirtualBox: Bridged Adapter


- Each VM requests its IP address just like the host OS to the default DHCP server
  - All VMs and host OS are in the same LAN, so they can talk to each other
  - Your home WiFi router most likely will support this

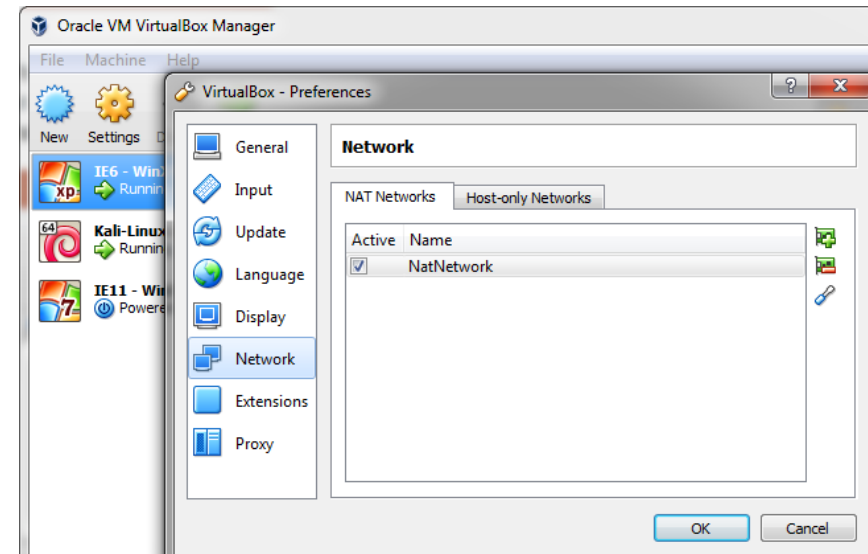


# Networking in VirtualBox: Bridged Adapter

- Problem: some DHCP servers do not provide service to VMs
  - WiFi does not provide IP to VMs
    - Your VM will not be able to obtain a valid IP
  - Your home WiFi router most likely will support this
    - Typically, you can use this networking setup at home.
    - But firewall rules may prevent the connections between clients even if they are within the same subnet.

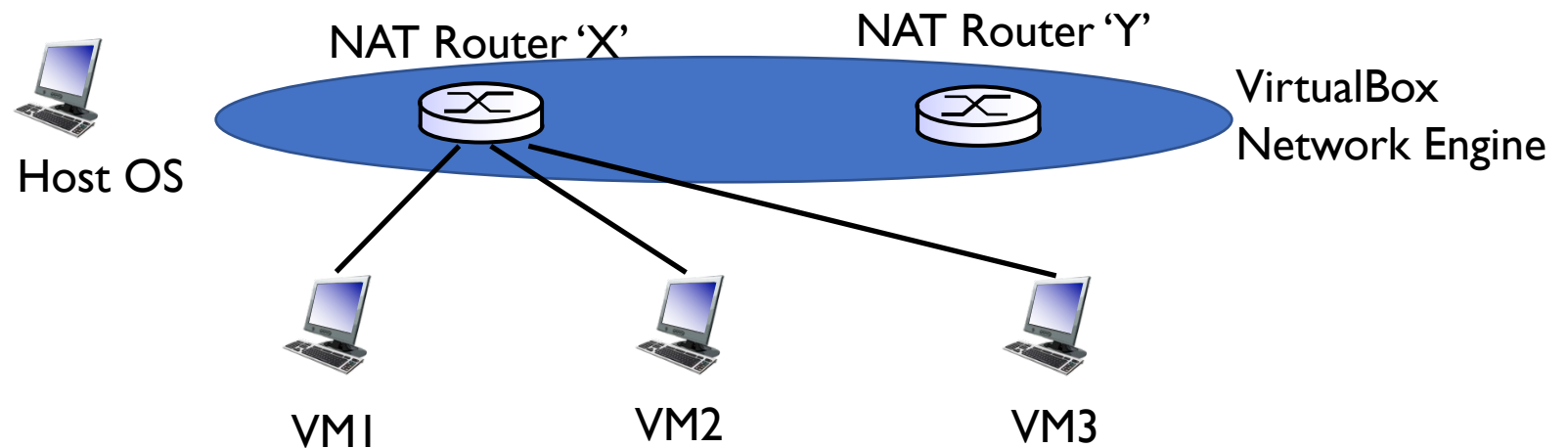
# VirtualBox Networking Option: NAT Network

- On VirtualBox, click “File” → “Preferences...” → “Network”
- If the “Net Networks” tab is empty, click  to add the default “NatNetwork”
  - You can change this NAT network name
- This will let VirtualBox to create a NAT router for Internal VMs that join in this NAT router



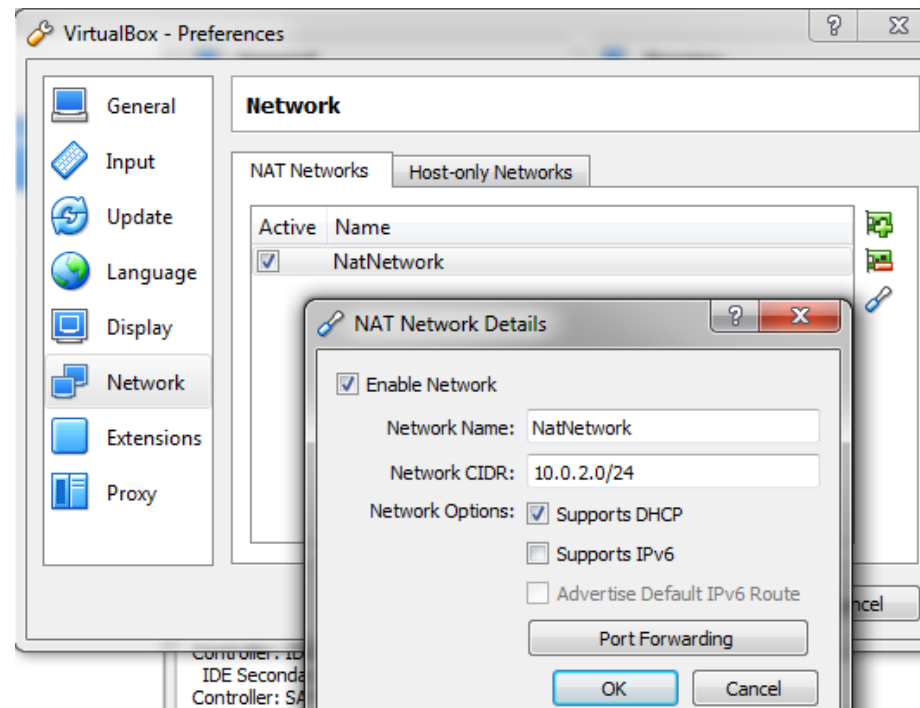
# Networking in VirtualBox: NAT Network

- VirtualBox setup a NAT router X
- All VMs join this NAT router X
- All VMs can see each other, in the same LAN
  - Host OS is not in this NAT router's LAN
- VirtualBox can set up multiple NAT Routers for multiple isolated VM LANs



# Networking in VirtualBox: NAT Network

- Determine local NAT LAN subnet:
  - Goto virtualBox menu: File→preferences...
  - On the NAT network, select the tool





# File Transfer between VM and Host OS under VirtualBox

## 1. Use online server for file upload/download

- Upload to an online storage (such as Google Drive, MS Onedrive)
- Download to the host OS or VM

## 2. Virtualbox support 'drag and drop' file transfer between host OS and a VM OS

- Run the Kali Linux VM under virtualBox
- Configure virtualBox menu "Devices" → "Drag and Drop" → enable "Bidirectional"
- In Kali, open "file folder" icon, in the host OS, open a folder window
- Now you can drag/drop files between host and VM

# Shared Folder in Linux VM

## 3. VirtualBox supports “shared folder” between host OS and VM

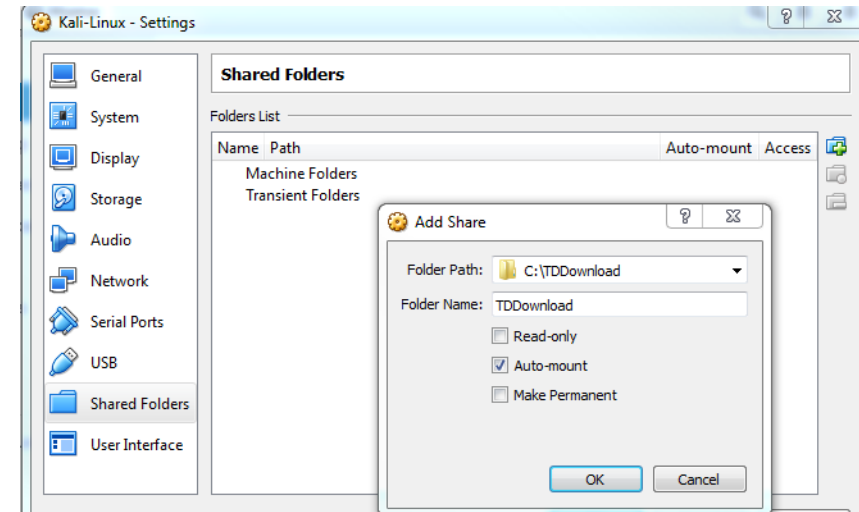
- Run the Kali Linux VM under VirtualBox
- Configure virtualBox menu “Devices” → “shared folders” → “Shared folder setting...” → click the “+” button
- In the Folder Path field, choose “Other...” to add a host OS folder as the shared folder (e.g., “Download”)

In Linux VM:

```
mkdir shared
```

```
mount -t vboxsf Download ~/shared
```

Now VM’s “~/shared” would be identical to the “Download” folder on host OS



# Shared Folder in Windows VM

- Configure virtualBox menu “Devices” → “shared folders” → “Shared folder setting...” → click the “+” button
- In the Folder Path field, choose “Other...” to add a host OS folder as the shared folder (e.g., “Download”)
- In Win VM, open folder, goto “network”, select “VBOXSVR”, then the shared folder will show up as a network drive

