INFINITYWORKS
ACADEMY

Generation

Source Control with Git

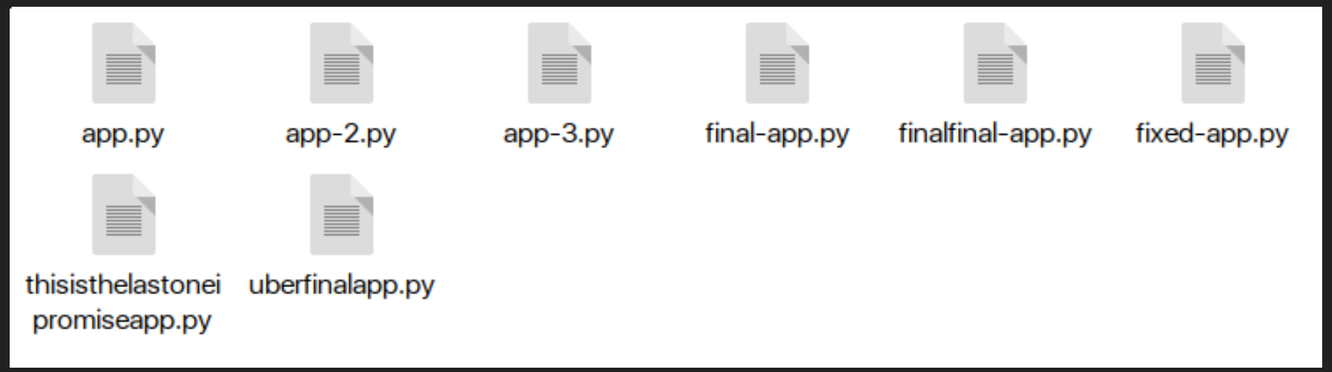
Overview

- Source Control
- Git
- Sharing Code
- Branching
- Merging
- Pull Requests
- Forking

Learning Objectives

- Describe what source control is, and why we use it
- Create a Git repository
- Create branches
- Add and commit features, and push to remote
- Create, review, and merge pull requests

A World Without Version Control



If we wanted to keep historical versions of a file, we would need to manually create a file for each version with a new name.

Keeping track of this would become cumbersome.

What is Source Control?

Source control allows you to keep track of your files and every change made to them, ever.

It also allows you to share these files and their change history with others, so they can make changes too.

Git

- A distributed source control management system (SCM)
- It has become the most widely used tool to manage source code nowadays

Exercise

Check you already have it installed:

```
$ git --version
git version 2.24.3 (Apple Git-128)
```

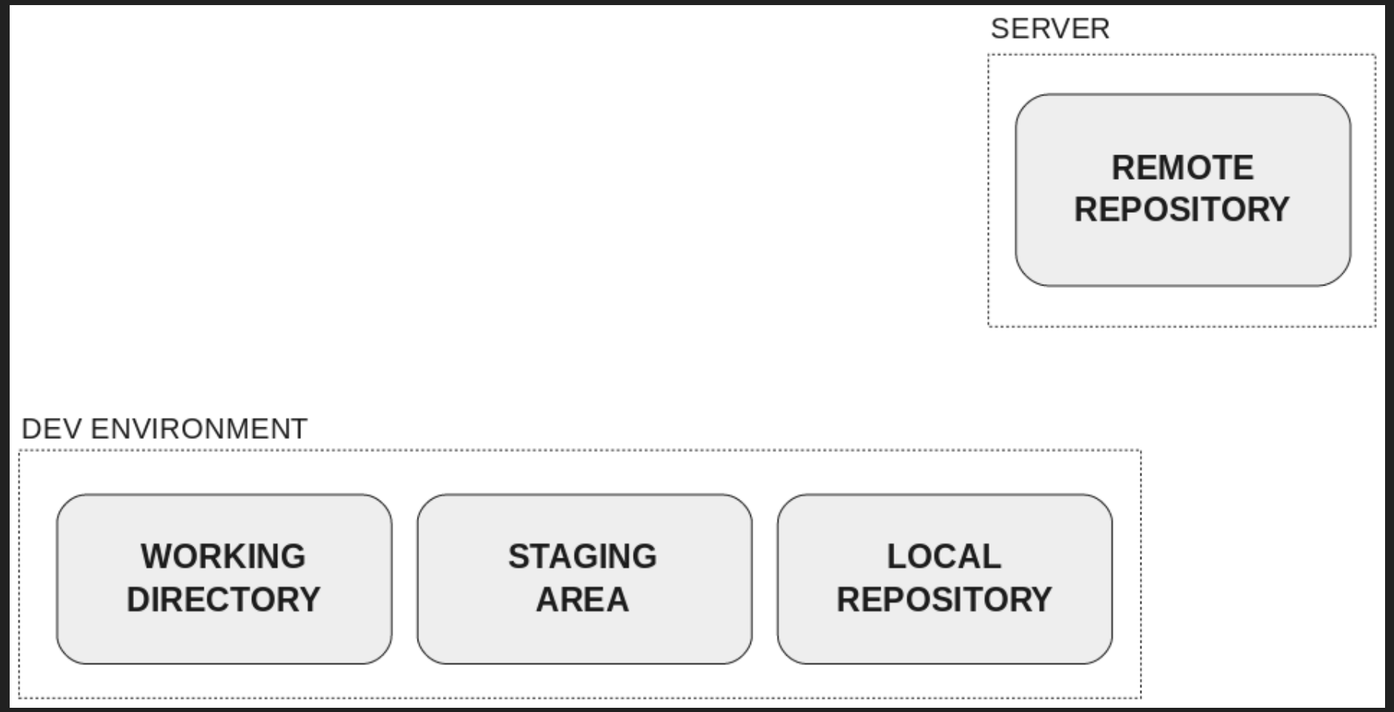
Install Git using your package manager if you haven't:

```
$ https://gitforwindows.org/ # Windows
$ sudo apt-get install git   # Unix
$ brew install git           # macOS
```

Configure Git on your machine:

```
git config --global user.name <your name>
git config --global user.email <your email>
```

Key Git components



Key Git components

Working directory: your private workspace, where changes are made. It may contain tracked and untracked files.

Staging area: all changes you want to bundle up in the next commit (transaction) to your local repository.

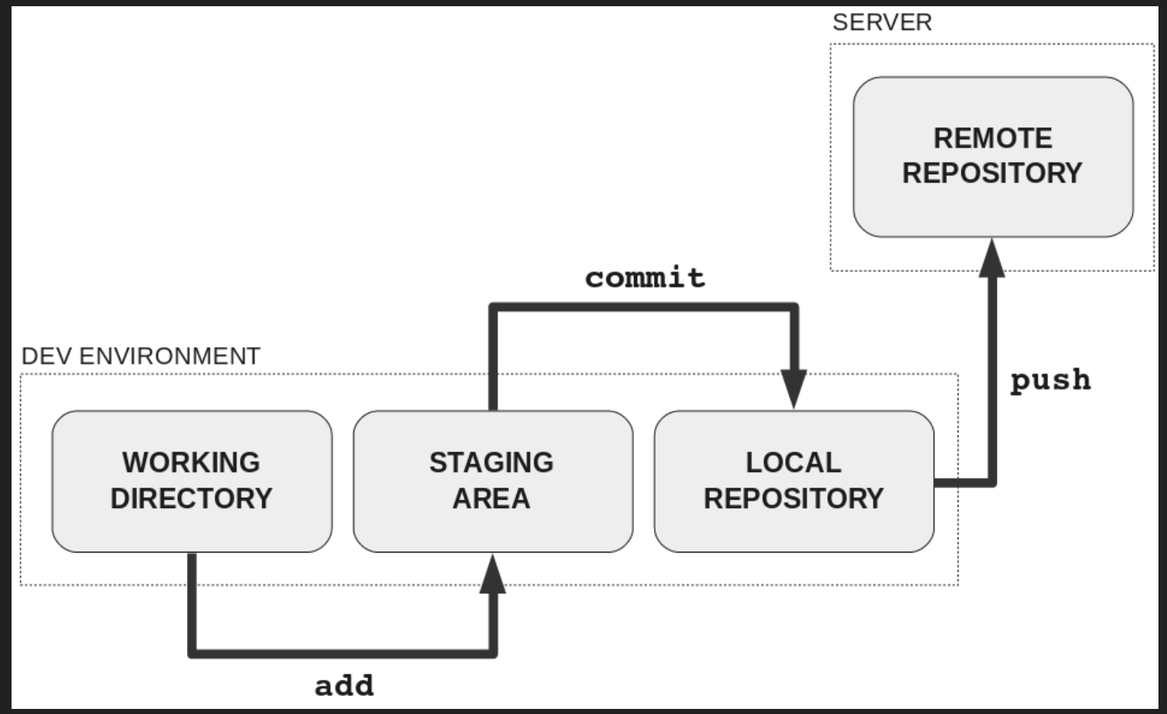
Local repository: your local version of the Git repository.

The Working Directory

Tracked: files that Git knows about

Untracked: files Git doesn't know about because they haven't been added yet

Making changes in Git



Making changes in Git

git add: make Git aware of a file (move to staging area)

```
git add myfile.py
```

git commit: apply change(s) to local repository

```
git commit -m "Update function definition"
```

git push: sync up your local repo with the remote repo

```
git push
```

git add

Mark files in your working directory as tracked so Git starts keeping a track record of all changes to them

If a file is already tracked and has some changes, move it to the staging area to include it in the next commit

```
# Stage one file
git add myfile.py
# Stage everything in the current directory (and subdirectories)
git add .
```

git commit

Bundle up all staged changes (or new files) in a new commit to apply to the local repository

```
# Write commit message in text editor
git commit
# Or inline commit message
git commit -m "Update function definition"
```

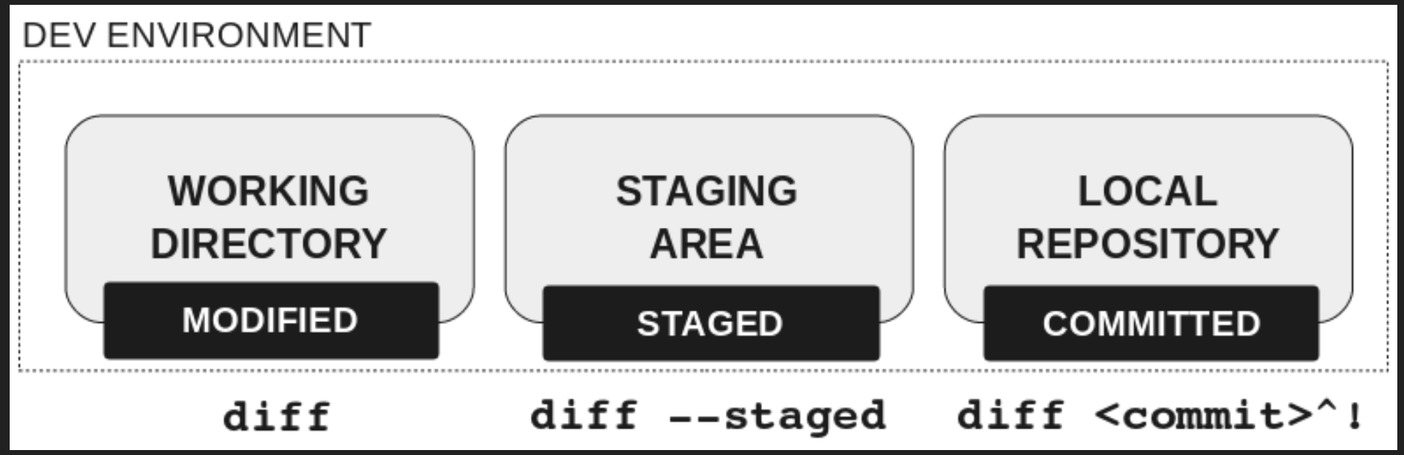
git push

Upload any changes made in your local repository to the remote repository:

```
git push
```

DANGER: if you pass in the -f flag, you will be rewriting Git history. Use with great caution.

Stages of a change



Example Git flow

```
git add myfile.py
git commit -m "Update function definition"
git push
```

Sharing Code

Sharing Git repositories

Git is distributed, which means most of the changes happen in people's individual repositories (repos) rather than in a central location.

Everyone's repos have all information about the repo.

There is no 'one and only' source of repo information.

Sharing Git repositories

Because Git is distributed, all information about a repository is copied every time it is cloned.

You upload your local changes to remote repositories hosted on private or public servers, and also download changes made by other people into yours.

Sharing Git repositories

Git repositories can be hosted in a private Git server, although this is something few people do. Most people prefer to utilise code hosting services like BitBucket, SourceForge, GitLab or GitHub.

GitHub

GitHub is the most popular code hosting website.

GitHub is free and offers loads of tools to aid coders:

- Code hosting
- Code discovery and search
- Bug reports
- Project task boards
- Project wikis

Quiz Time! 🤓

Which git command would you use to apply your local changes to your local repository?

1. `git status`
2. `git add`
3. `git commit`
4. `git push`

Answer: `3`

Exercise

- Create a GitHub Account
- Create a new GitHub repository for your mini-project under your account
- Set up an SSH key for GitHub
  - Follow the *Generating a new SSH key* instructions here
  - Then follow the *Adding a new SSH key to your GitHub account* instructions [here](https://docs.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account
  - Setup Multi-Factor Authentication (MFA) on your Github Account

We now have a repository set up on GitHub for our mini-project code 🎉
How do we actually get our code from our machine up to our GitHub repo?

Exercise

Switch to the top-level directory where your mini-project code is located, and run:

```
# this creates a local repository for our code
git init
```

Then run:

```
# this sets the repository living at that URL as your remote
git remote add origin git@github.com:<user>/<repository>.git

# this checks that the remote was added correctly
git remote -v
```

git status

`git status` gets the status of your dev environment. It displays information about:

- Untracked files
- Uncommitted changes
- Whether your local repo needs syncing with the remote

Exercise

- Check the status of your dev environment
- Stage all of your mini-project code so far
- Commit your code with the message "Initial commit"
- Push your mini-project code to your remote repository

Git History

Every commit in your git history is given a unique ID known as a hash.

We refer to individual commits by their hashes. The last commit in your repo is known as the HEAD.

Rolling back changes

We can roll back our repo to any past commit in the history.

The HEAD pointer contains the last commit in our repo by default but we can change that, thus allowing us to go back to any point in the past of our repo

```
$ git reset < commit_hash >
# The --hard flag will reset the HEAD pointer and delete all newer commits
# Use with caution!
$ git reset --hard < commit_hash >
```

Use `git log` to look at your history.

```
| | * | | | | | c3c9e3df49f8 - Mon, 19 Aug 2019 09:25:37 +0100 (3 weeks ago)
| | | | | | |          rxrpc: Improve jumbo packet counting - David Howells
| * | | | | | | 3b25528e1e35 - Thu, 29 Aug 2019 11:17:24 +0800 (11 days ago)
| | | | | | |          net: stmmac: dwmac-rk: Don't fail if phy regulator is absent - Chen-Yu Tsai
| * | | | | | | b6b4dc4c1fa7 - Thu, 29 Aug 2019 10:46:00 +0800 (11 days ago)
| | | | | | |          amd-xgbe: Fix error path in xgbe_mod_init() - YueHaibing
| * | | | | |   869326532956 - Thu, 29 Aug 2019 16:44:15 -0700 (10 days ago)
| |\ \ \ \ \ \            Merge tag 'mac80211-for-davem-2019-08-29' of git://git.kernel.org/pub/scm/linux/kernel/git/jberg/mac80211 - David S. Miller
| | * | | | | | f8b43c5cf4b6 - Tue, 27 Aug 2019 17:41:20 -0500 (12 days ago)
| | | | | | |          mac80211: Correctly set noencrypt for PAE frames - Denis Kenzior
| | * | | | | | c8a41c6afa27 - Tue, 27 Aug 2019 17:41:19 -0500 (12 days ago)
| | | | | | |          mac80211: Don't memset RXCB prior to PAE intercept - Denis Kenzior
| | * | | | | | b9500577d361 - Wed, 21 Aug 2019 20:17:32 +0300 (3 weeks ago)
| |/ / / / / /           iwlwifi: pcie: handle switching killer Qu B0 NICs to C0 - Luca Coelho
| * | | | | | 189308d5823a - Wed, 28 Aug 2019 08:31:19 +0200 (11 days ago)
| | | | | | |          sky2: Disable MSI on yet another ASUS boards (P6Xxxx) - Takashi Iwai
| * | | | | |   807e32999567 - Wed, 28 Aug 2019 16:06:49 -0700 (11 days ago)
| |\ \ \ \ \ \            Merge branch 'nfp-flower-fix-bugs-in-merge-tunnel-encap-code' - David S. Miller
| | * | | | | | e8024cb483ab - Tue, 27 Aug 2019 22:56:30 -0700 (11 days ago)
| | | | | | | |          nfp: flower: handle neighbour events on internal ports - John Hurley
| | * | | | | | 739d7c5752b2 - Tue, 27 Aug 2019 22:56:29 -0700 (11 days ago)
| |/ / / / / /           nfp: flower: prevent ingress block binds on internal ports - John Hurley
| * | | | | |   80a6a5d62da9 - Wed, 28 Aug 2019 16:02:32 -0700 (11 days ago)
| |\ \ \ \ \ \            Merge branch 'r8152-fix-side-effect' - David S. Miller
| | * | | | | | 973dc6cfc0e2 - Wed, 28 Aug 2019 09:51:42 +0800 (12 days ago)
| | | | | | | |          r8152: remove calling netif_napi_del - Hayes Wang
| | * | | | | | 49d4b14113ca - Wed, 28 Aug 2019 09:51:41 +0800 (12 days ago)
| |/ / / / / /           Revert "r8152: napi hangup fix after disconnect" - Hayes Wang
| * | | | | | 092e22e58623 - Tue, 27 Aug 2019 23:18:53 +0200 (12 days ago)
| | | | | | |          net/sched: pfifo_fast: fix wrong dereference in pfifo_fast_enqueue - Davide Caratti
| * | | | | | 888a5c53c0d8 - Tue, 27 Aug 2019 15:09:33 -0400 (12 days ago)
| | | | | | |          tcp: inherit timestamp on mtu probe - Willem de Bruijn
| * | | | | | dbf47a2a094e - Tue, 27 Aug 2019 21:49:38 +0300 (12 days ago)
```

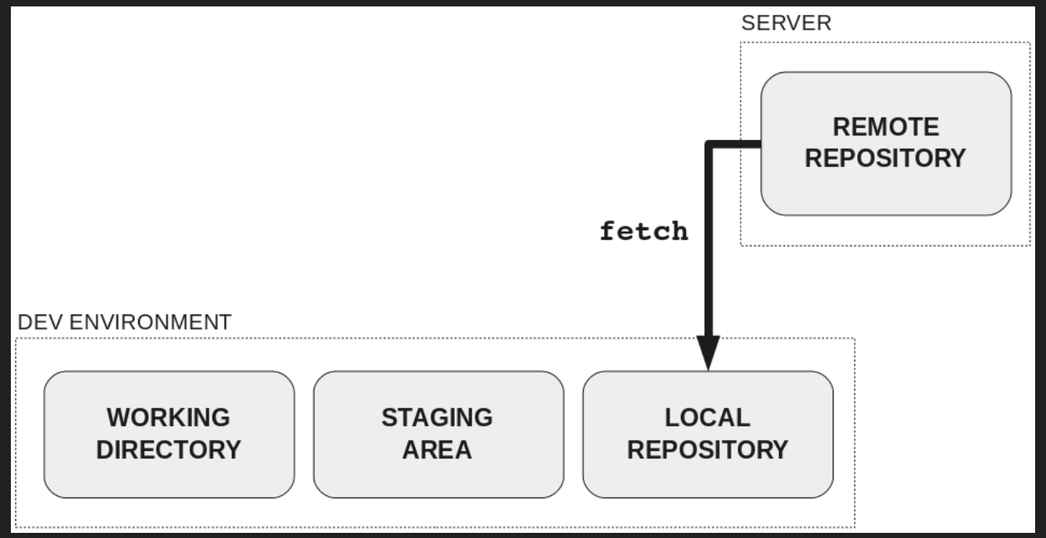
Downloading an existing repo

Instructor to provide unique GitHub URL.

```
git clone <url>
```

Updating the dev environment: fetching

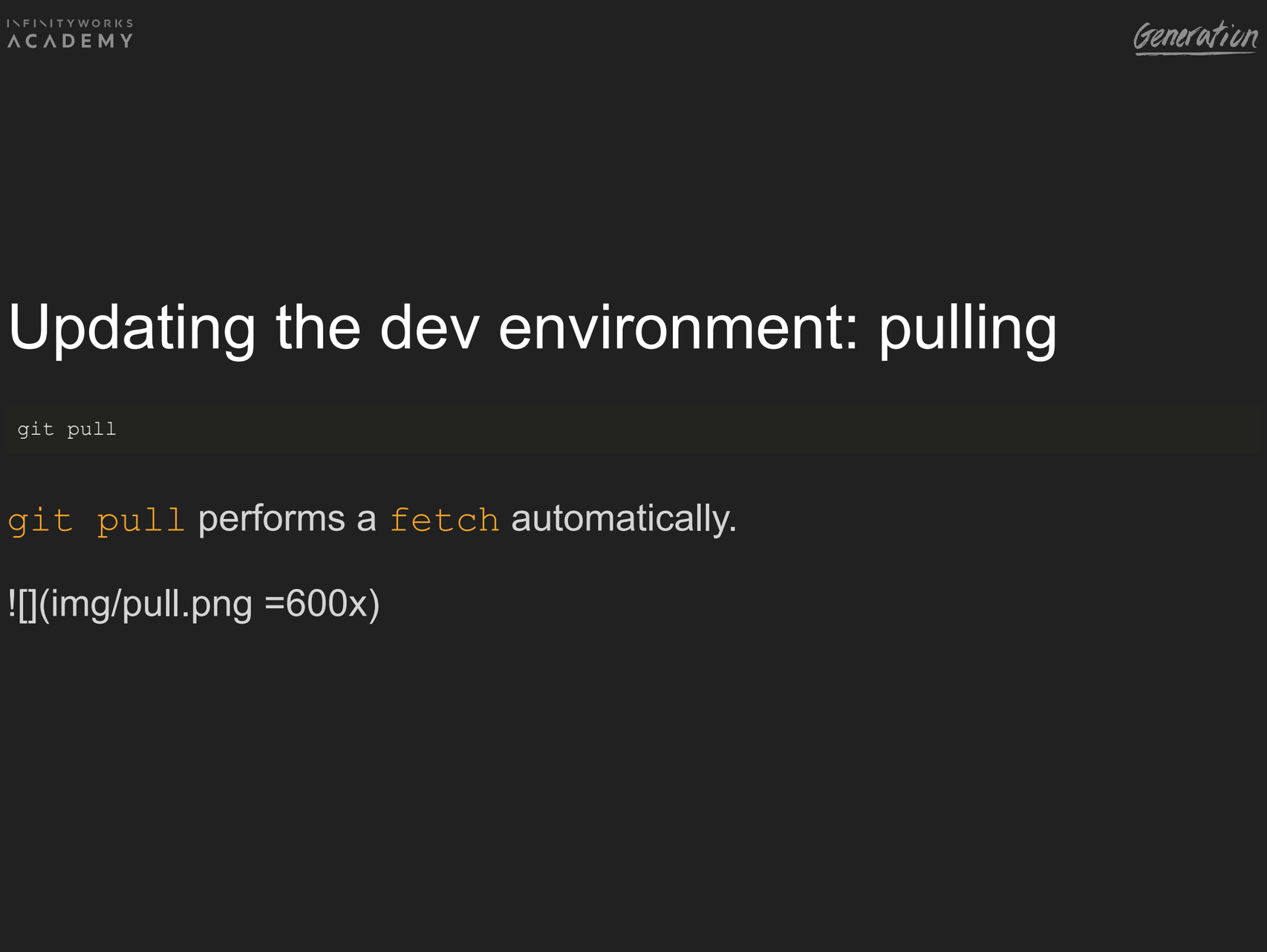
```
git fetch
```



Updating the dev environment: pulling

```
git pull
```

`git pull` performs a `fetch` automatically.



Branching

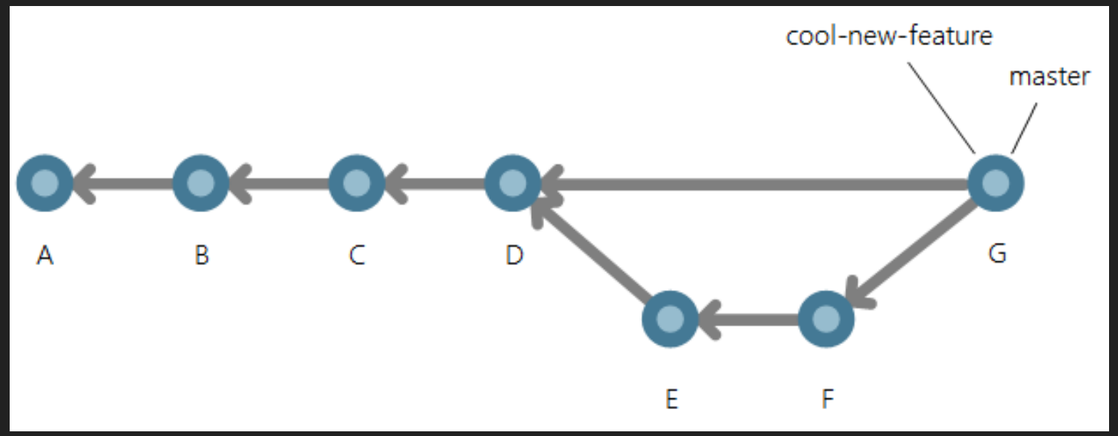
What is a branch?

A branch is an alternative development path in your repository.

Branches are created on top of a commit in another branch (usually master).

Branches append new commits on top of the commit they stand on, without affecting the original branch.

What is a branch?



Creating and checking out branches

```
git branch < new_branch >
git checkout new_branch
```

```
# You can create and switch to a branch in one go
git checkout -b new_branch
```

Pushing branches

`--set-upstream` creates an upstream branch in the remote that matches our local branch of the same name.

```
git push --set-upstream origin new_branch
git push -u origin new_branch
git push -u origin HEAD
```

Not necessary if pushing to `main`, as that's the default.

Scenario

Alice has written some code on a feature branch and pushed her branch up to the remote. You want to have a look at her code and try it out for yourself. How can you do this?

1. `git branch alices-branch`
2. `git fetch && git branch alices-branch`
3. `git checkout alices-branch`
4. `git fetch && git checkout alices-branch`

Merging

Once work in a branch is done, we can merge a branch back into the branch it was splintered off from so that our changes can be applied to it.

```
git merge new_branch
```

Merge types: fast-forward

If there are no conflicts and no changes have been made to the current branch, a fast-forward will occur.

The HEAD pointer will be moved forward to the latest commit that we pulled.

Merge types: merge

If there are changes in the current branch but not in the same areas as the branch we're merging, Git will attempt to automatically merge these changes for us.

Merge conflicts

If the current branch was modified in the same places as the branch we're merging, there may be conflicts which Git can't resolve on its own.

Scenario

Bob has added a fix to his branch which would be helpful to the code you're currently working on. What command should you use to get Bob's changes into your current branch?

1. `git branch bobs-branch`
2. `git fetch; git merge bobs-branch`
3. `git pull origin bobs-branch`

Demo - interacting with Git

1. Create a new branch on the repository you just cloned
2. Navigate to `about-me.txt` and follow the instructions
3. Add and commit your changes
4. Push your work up on the new branch
5. See your changes on GitHub!

Pull requests

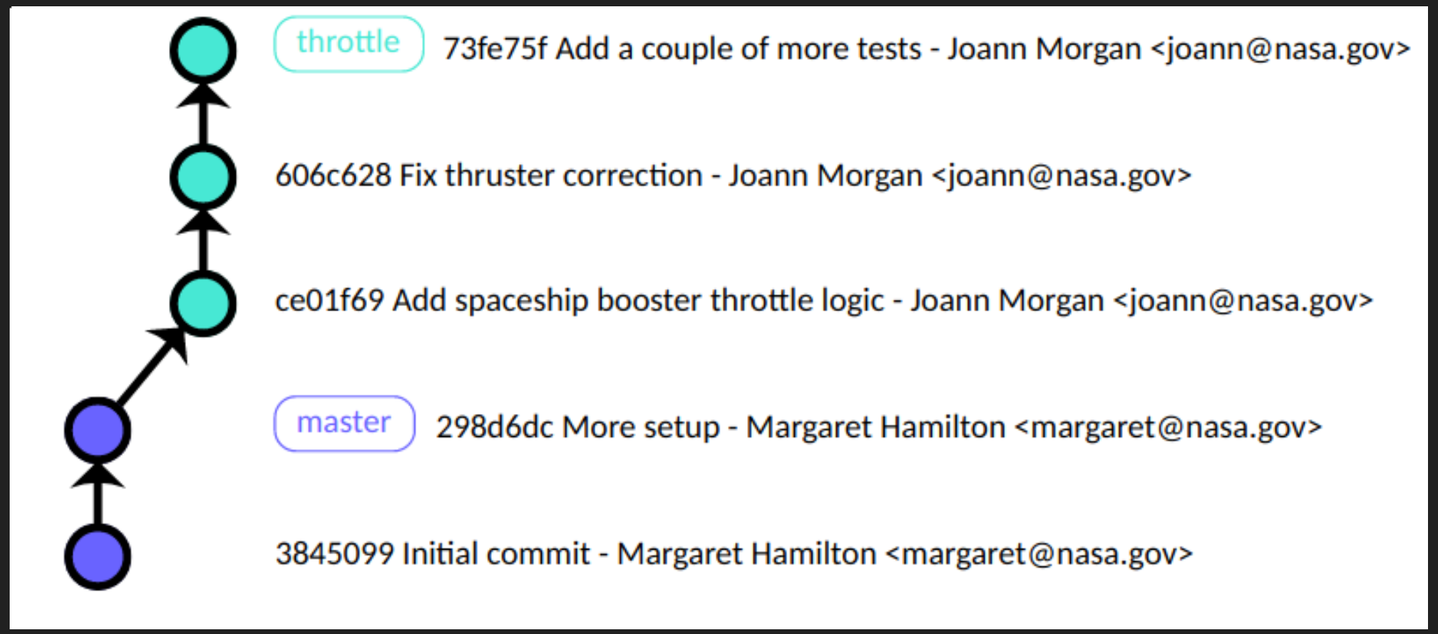
Pull requests (AKA merge requests) allow us to control changes going into our `main` branch.

When you raise a PR, you're asking another developer to pull your branch into their own repository.

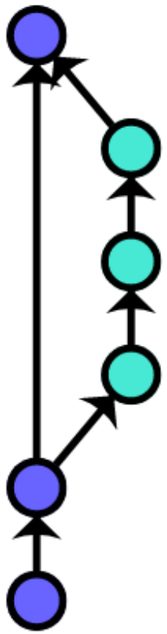
The PR can then be reviewed and tested and if it's accepted, merged into a branch in their own repo.

Example

Joann creates a new branch, based on top of `main's` last commit. She then proceeds to create commits on this branch.



Example



master ad1ed6d Merge branch throttle - Margaret Hamilton <margaret@nasa.gov>

throttle 2ef824a Add a couple of more tests - Joann Morgan <joann@nasa.gov>

702da56 Fix thruster correction - Joann Morgan <joann@nasa.gov>

097ed8a Add spaceship booster throttle logic - Joann Morgan <joann@nasa.gov>

641dd3f More setup - Margaret Hamilton <margaret@nasa.gov>

ff39a9d Initial commit - Margaret Hamilton <margaret@nasa.gov>

Demo - creating a PR

1. Create a Pull Request for your branch
2. Attempt to merge your change
3. Resolve merge conflicts if required (you hopefully won't get any)

Forking

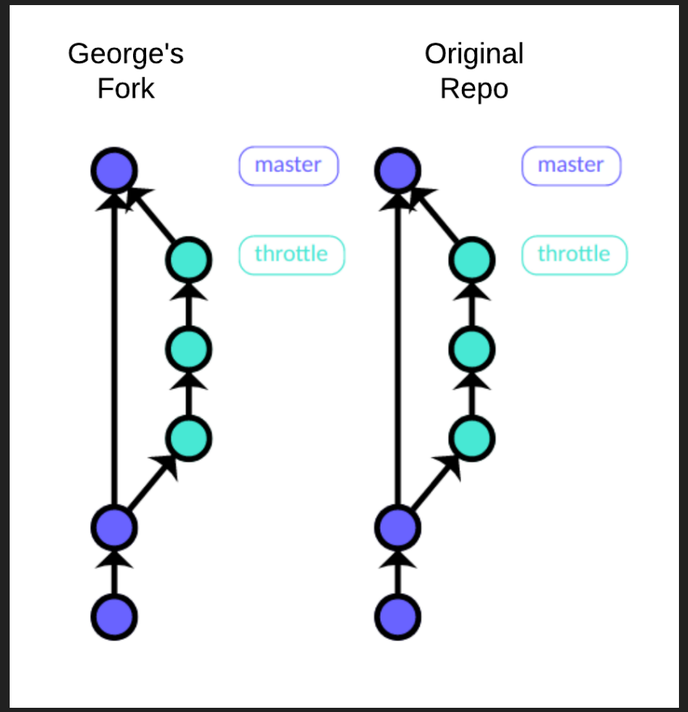
Forking a repo means creating a verbatim copy of it, which we control.

A fork is an exact copy of the original, including source, commit history, refs and everything else!

Any changes we make to our fork will not affect the repo it was copied from.

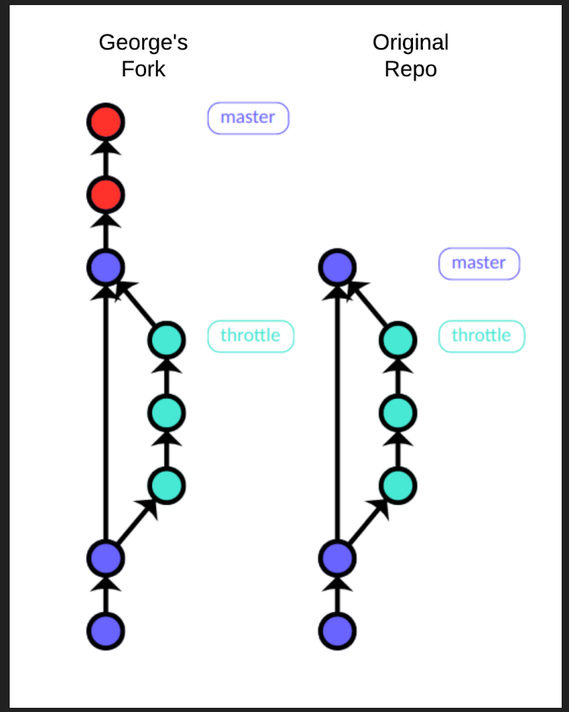
How do we incorporate our changes into the mainline repo?

Example

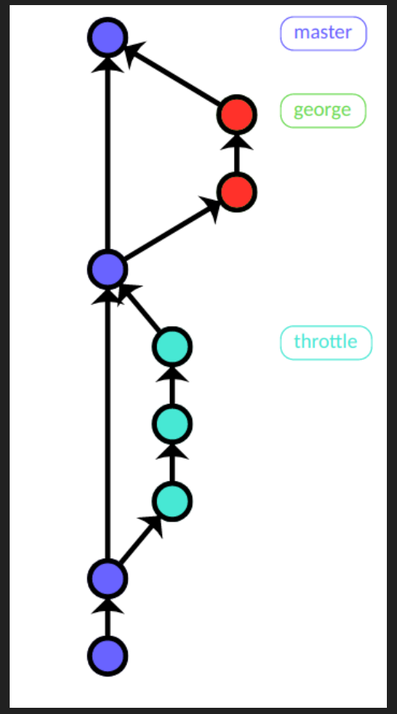

George's Fork
Original Repo
master
throttle
master
throttle

George has made two new commits in his repo and now needs to raise a PR to merge them into the original.

Example


George's Fork
master
throttle
Original Repo
master
throttle

Example



Commit etiquette

Commits should:

- Pass all tests, and not have any known bugs to be fixed in a future commit.
- Be comprised of self-contained changes.
- Have short and meaningful commit messages.

Example commit message: *change function to return integer value rather than string value*

Other useful Git commands

git stash

The `stash` is a stack you push changes to that you want to undo temporarily.

You don't want to scrap your work, but you're not ready to commit it just yet either.

```
# Shelf your current changes - they disappear from your tree
git stash
# Retrieve the last set of changes you saved
git stash pop
```

git blame

Find out the author of a specific commit.

```
git blame < commit_hash >
```

git show

Show changes introduced in a commit.

```
git show < commit_hash >
```

.gitignore

Include this file in your project to tell git to ignore certain files or patterns so they're never tracked

https://github.com/github/gitignore

One final important message

NEVER commit sensitive information! This includes:

- Passwords
- Keys
- Private information about a person or entity

You can use tools such as git secrets to help prevent yourself from committing them.

Quiz Time! 🤓

What order of Git commands would you use to (1) check what local changes you've made, (2) get these changes tracked, (3) apply them to the local repository, and (4) apply them to the remote repository?

Let's see how much you can remember.

1. `add, status, commit, push`
2. `status, add, commit, push`
3. `status, commit, push, add`
4. `add, commit, status, push`

Answer: `2`

Exercise

- Write a README document for your application in Markdown. Document how to set up, run, and contribute code to your app. Push your changes.
- Clone the repo of a partner, identify a *small* code smell, refactor it and raise a PR for it on GitHub
- Do a code review on GitHub for the PR raised on your own project
- Merge it if it meets your strict quality standards

Learning Objectives Revisited

- Describe what source control is, and why we use it
- Create a Git repository
- Create branches
- Add and commit features, and push to remote
- Create, review, and merge pull requests

Terms and Definitions Recap

Repository: Stores all your code, and all history and tracking metadata

Commit: A number of changes bundled up as a single transaction

Branch: Alternative code path originating from a specific commit in another branch

Staging: Telling Git what changes we want to include in the next commit

Terms and Definitions Recap

Clone: Copy a remote Git repository and all of its metadata into our local environment

Fork (GitHub): Make a copy of a repo hosted online into another hosted repo under our account so we can freely change it

Pull Request / Merge Request: Request to merge a branch/fork into the main branch/fork it originated from (upstream)

Further Reading

- Pro Git Book (free)
- git tutorials
- Pull requests tutorial
- Credits to Rachel Carmena
- Great introductory article to git
- `man git`