Git For Everyday Development

Grand Valley State University
September 26, 2019

ATOMIC OBJECT

Who We Are

We’re software developers and members of the Accelerator program at Atomic Object.

Atomic Object creates applications for web, mobile, desktop, and devices. We help companies innovate and grow with custom software products that are beautiful, reliable, and easy to use. Offices in Grand Rapids and Ann Arbor.

Read more about the Accelerator program here.

Workshop Goals:

- You'll be comfortable using basic Git commands.
- You'll be able to use Git for school and personal projects.
- You'll have a basic understanding and mental model of how Git works.
- You'll have a good idea of where you can go to learn more about Git.

What is Git?

Git is a version control system that keeps track of how files in a codebase have changed.

Git tracks things like what changed in a file, who changed it, and their reason for making that change.

What is Git?

Git allows others to work in the same repository (project) as you, so you can share your progress with them.

Git is the most widely used version control system in the world, and the majority of software developers use it every day.

Setup

Setup

Clone the repository to your local machine from GitHub using your terminal:

```
$ git clone <github-url>
```

Navigate to the repository on the command line:

```
$ cd <path-to-repository>
```

Open the webpage and look around:

```
$ open index.html
```

Need a command line refresher? Reference the Git cheat sheet or raise your hand.

Making Your First Commit

Making Your First Commit

Check which files have been changed since your last commit:

```
$ git status
```

Check what changes have been made line by line:

```
$ git diff
```

Tip: Use `git diff <filename>` to see the changes for a single file.

Making Your First Commit

Stage the changed files that you would like to commit:

```
$ git add <file(s)>
```

or

```
$ git add --all
```

Commit all staged files with a commit message:

```
$ git commit -m “<message>”
```

Take a look at the reference manual for more information on commit message best practices.

Making Your First Commit

Push to a remote repository:

```
$ git push
```

Making Your First Commit

Open the `index.html` file in your chosen text editor.

Change the placeholders for name and school in the "About Me" section.

Use `git status` to check which files have changed.

Add your changes and commit them.

Push your commit up to GitHub.

Making Your First Commit

We made changes directly on `develop`, but this isn't usually how you want to do things.

Each feature should have its own branch so that your changes don't affect the main repository.

We'll talk about branches and merging next.

Creating Branches

Creating Branches

Create a new branch:

```
$ git checkout -b <branch-name>
```

Checkout a branch:

```
$ git checkout <branch-name>
```

Go back to the previous branch:

```
$ git checkout -
```

Typically, branch names start with the type of work you're doing and a slash, like `feature/add-new-button`, or `bug/fix-login-issue`.

Creating Branches

Create a new branch off of `develop` for your first task. Name it what you want, but we'll call it `branch-1` for the rest of the workshop.

Go back to `develop`.

Create a new branch off of `develop` for your second task. We'll call it `branch-2`.

Implementing a Feature

Implementing a Feature

Merging another branch into your working branch:

```
$ git merge <other-branch>
```

Set an upstream branch:

```
$ git push -u origin <branch-name>
```

Implementing a Feature

Checkout `branch-1`.

Find the HTML for the GVSU image and wrap it in a link that goes to the Grand Valley site.

Add, commit, and push your changes when you finish.

Checkout `develop` and merge `branch-1`.

You'll need to set an upstream when you push from a new branch for the first time.

Implementing a Feature

Go to `branch-2`.

Create a `div` element that wraps both images, and set the background color of the `div` to a color of your choice.

Add, commit, and push the changes you just made.

Handling Merge Conflicts

Handling Merge Conflicts

This is how you know there was a merge conflict:

```
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

This is what a merge conflict will look like in your code:

Handling Merge Conflicts

Make sure you're on `branch-2`.

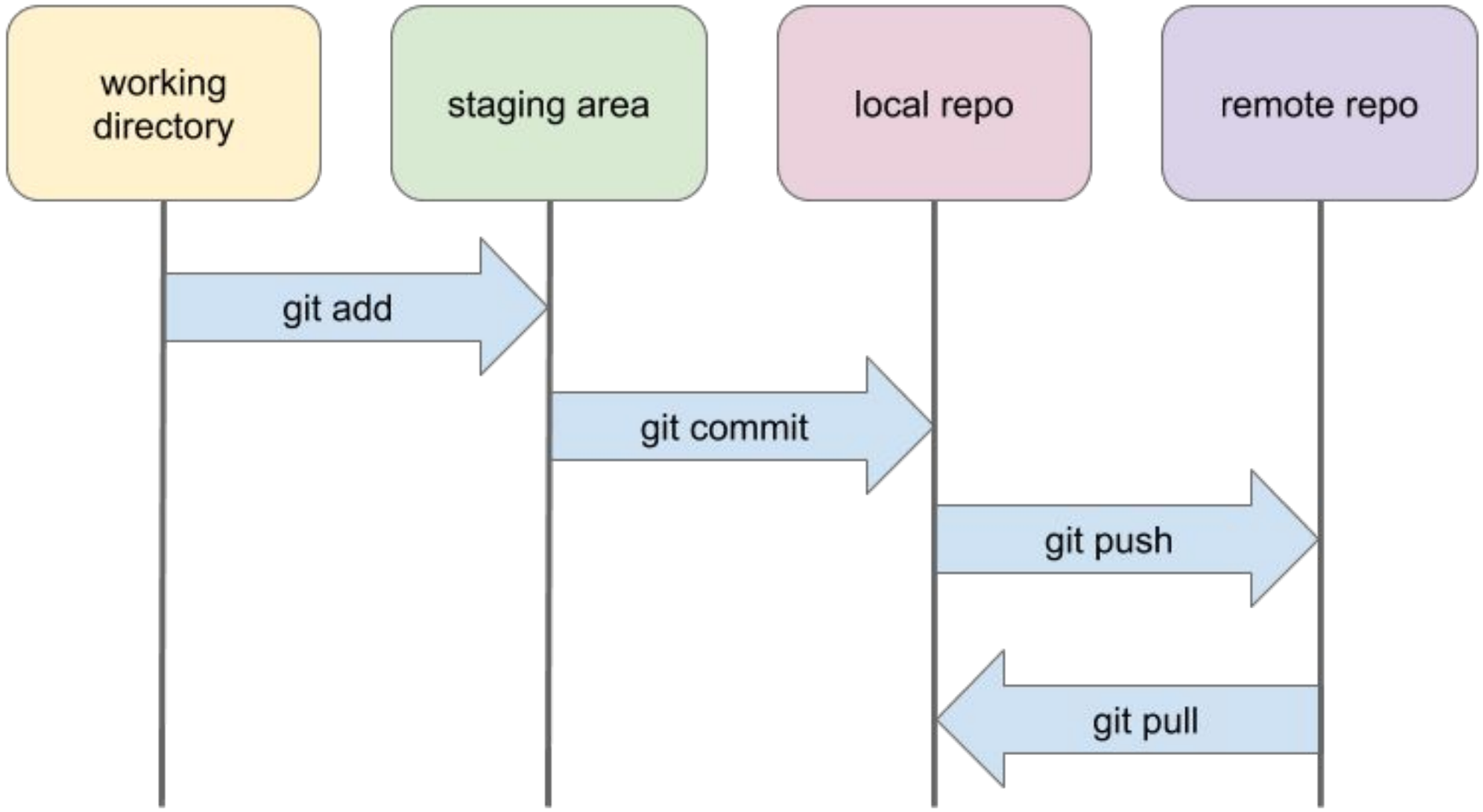
Merge `develop` into `branch-2`. You should get a merge conflict.

Once you've resolved the conflict, merge `branch-2` into `develop`.

Use `git status` if you're unsure which branch you're on.

The Git Model

The Git Model



A remote repository is the actual main copy of the repository that exists wherever the repo is hosted (GitHub, in this instance).

Upcoming Student Opportunities

Atomic Games - October 26 - Apply below:

https://atomic-games.atomicobject.com/

Student Open House - October 10, 11 - RSVP below:

http://atomobj.io/openHouse2019

Atomic Accelerator Program Information:

https://atomicobject.com/careers/accelerator


ATOMIC OBJECT

Sources

Git Workflow Diagram Based On:

https://tex.stackexchange.com/questions/70320/workflow-diagram?rq=1

Sample HTML Page:

https://html5-templates.com/preview/bootstrap-scrolling-sticky-menu.html