Introduction to Git

—

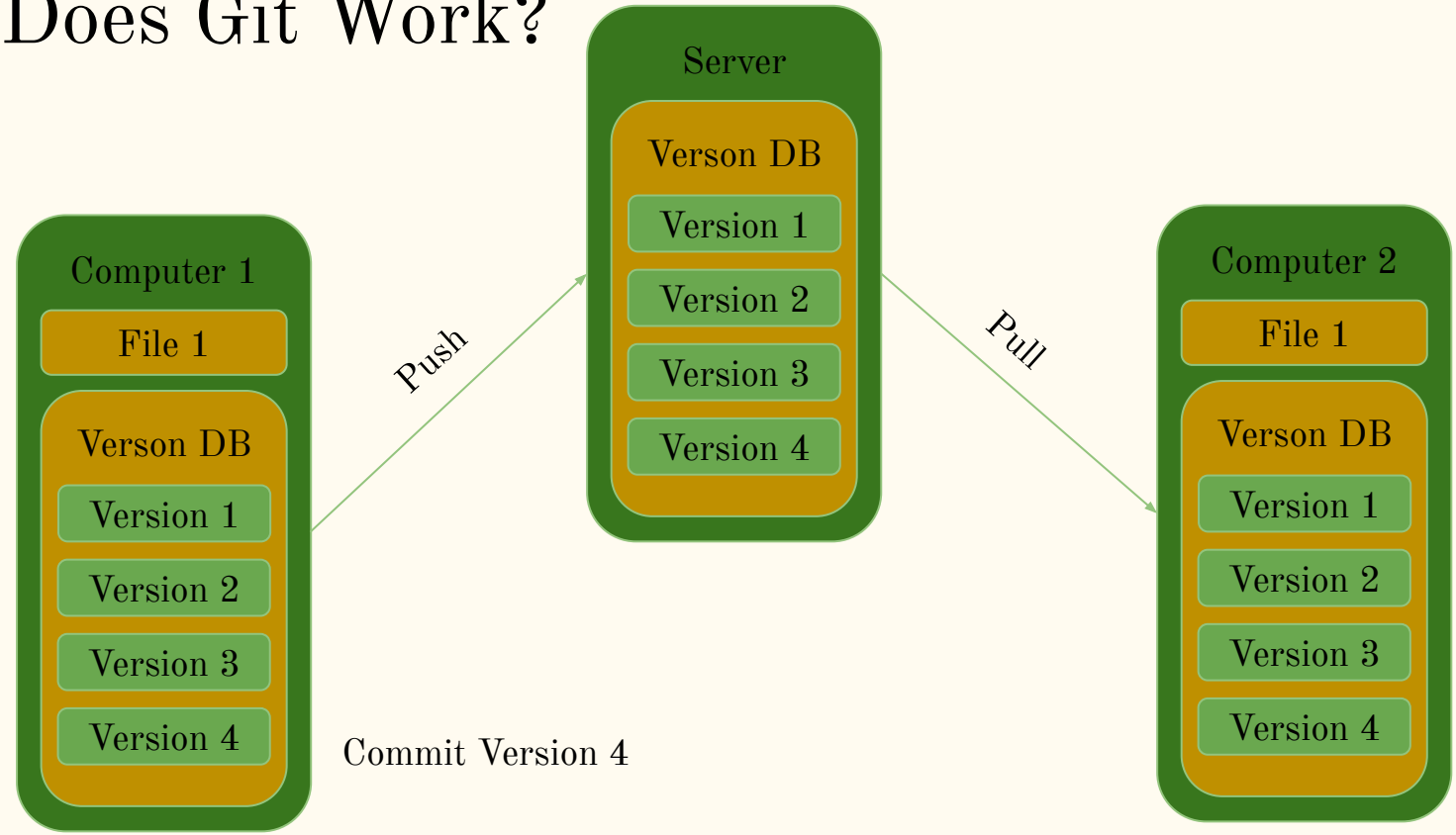
Game With Us Game Jam

By: Daniel McCoshen and Max Prosak

What is Git?

- Git is a decentralized version control system.
- What's it good for?
  - Keeping track of changes.
  - Allowing multiple people to work on the same project and files.
  - Separating work on different parts of a project
  - Keeping a stable version of your code separate from current work

How Does Git Work?



Git Basics

Git <command> <arguments>

- Config
  -
- init
- clone <repository>
  - Clones a repository into a newly created directory.
- pull
- checkout <branch>
  - Changes between branches.
  - Make sure you pull to ensure it’s up to date.
- fetch
- commit

Getting Started

- Two common scenarios
  - Creating an empty repo on Github, or a similar service
    - Easy!
  - Create a repo from existing code
    - Harder

Starting from Github (or similar)

- On Github
  - Click 

 from the top right of the dashboard
  - Enter repo name, make it descriptive
  - Optionally add a description
  - Choose public or private
  - Check off 

  - Click 

- On your computer
  - git clone <your repo url>
    - eg. https://github.com/DanielMcCoshen/gamejam2017try2.git
      - This is one of Dan's past game jam games
    - Repo can be cloned through https or ssh, but ssh requires additional setup

Starting from Existing Code

- On Github
  - Do what you just did, but without checking off 

- On your local machine
  - $ cd /path/to/code
  - $ git init
    - This creates a .git subdirectory containing repo information.
  - Create a .gitignore
    - Optional, more on this later
  - $ git add -A
    - Set all files to be tracked, and stage them for their first commit
  - $ git commit -m “initial commit”
    - commits all files with the message “initial commit”

Starting from Existing Code

- $ git remote add origin <your repo url>
  - This sets up your repo to track the remote changes in the repo
- $ git push -u origin master
  - This pushes your committed code to github
- If you check your repo on github, you should now see the files you commited

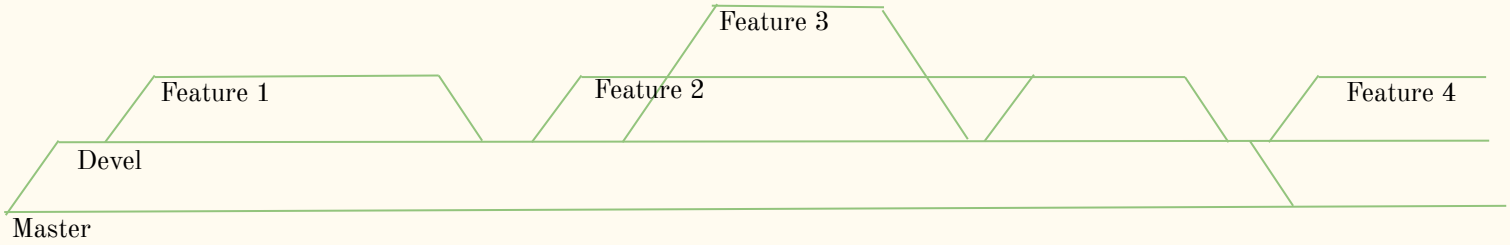
Staging vs Committing

- Git has a feature where changed files can be staged for commit separately
  - Useful for splitting work into multiple commits
  - Can be confusing for new users
- Stage files with git add
  - $ git add -A
    - stages all changes and untracked files
  - $ git add <file>
    - Stages changed or untracked file
- If you don't want to break changes into multiple commits
  - $git commit -a
    - Stages and commits all changes
    - A bit a wide net, can sometimes include things you would rather not include

Branching and workflow

- Git allows multiple code branches to exist
  - Great for isolating work on multiple features or parts of the project
  - Can break things if not carefully done
    - Eg. if you merge willy-nilly you can lose work
- When a repo is created, the default branch is ‘master’
  - This should contain your most recent completely stable code, and nothing that is in progress
- Create a separate branch for developing on.
  - Call it ‘devel’, ‘develop’ or something similar
  - This is code that isn't fully tested or stable, but isn’t currently being written
- Each new feature should have its own branch, branched off of develop
  - Name them something like ‘feature/character_movement’
  - This is code that is actively being written

Branching Diagram



Branching Considerations

- When you make a branch, it is branched off of your current branch at the current commit
  - Make sure you are on the right branch before creating a new branch
    - Don’t accidentally branch a new feature off of an old feature
  - Make sure your branch is up-to-date before branching
    - $ git pull
      - to pull new changes
    - If you don’t you might be missing code critical to what you are working on
      - Recoverable, just change branch to devel, pull, change branch to feature, merge devel

Branching Considerations

- Fix merge conflicts on your feature branch not develop
  - Before merging a feature branch into develop, merge develop into your current branch
  - $ git merge develop
  - If there are merge conflicts, edit the conflicting files, then git commit -a
  - Test your code, in case you broke something. It happens
  - Then merge to develop
- Use Pull Requests on Github, rather than merging on your computer
  - It allows you to have code reviewers
  - Won't let you merge with conflicts
  - Helps keep a record of what’s been done

Useful Branching Commands

- $ git branch
  - Branch management
  - $ git branch <branch name>
    - Create a new branch
  - $ git branch -d <branch name>
    - Delete a branch
  - $ git branch -m <new branch name>
    - Rename your current branch
    - Does weird stuff if this branch has already been pushed to the server
- $ git checkout
  - $ git checkout <branch name>
    - Switch which branch you are on
  - $ git checkout -b <branch name>
    - Create a new branch and switch to it

.gitignore Files Make Your Life Better

- Create a file in your repository root called .gitignore
- In this file specify file/directory name patterns that git should ignore
  - It will act like these files don't exist whatsoever
- Use this to ignore compiled files, IDE and Unity settings, and other things like this that should not be included in your repo
- Depending on your game engine, there will be different files to include in this
- Unity has released a generic one for unity projects
  - Use it.
    - Seriously. Use it
  - We recommend adding .DS_Store files to it if you intend on developing on Mac
- Github has many premade gitignore files, here is a url to Unity's https://github.com/github/gitignore/blob/master/Unity.gitignore