

Introduction to Git & GitHub

Presented and prepared by Simeon Ramjit



Back it* up, like a hard-drive

-Shenseea, singer-songwriter

* 'It' refers strictly to data

What is Git?

- Distributed Version Control System 🎉
 - Records file changes
 - Rollback files to any point in time
 - Easily collaborate with other developers on the same project

Why Git?

- Managing project files can be difficult
- Tracking changes manually is a task in itself
- Storing versions of projects on your disk is impractical at scale
 - Just imagine something like this:

```
yourawesomeproject
├── awesomeproject v1
├── awesomeproject v2
├── awesomeproject v3
├── .....
└── awesomeproject v30.x.xx
```

Why should we use Git?

- It handles tracking changes and who's responsible for those changes
- Branches allow you to work on modular features without working on the main code
- Collaboration and accountability are simplified

Git vs GitHub

Git – Version Control System

- Local – installed and maintained on your machine
- Uses command line

GitHub – Source code management

- Cloud hosted – designed to host Git repositories
- Uses a GUI
- Has additional functionality like fork, pull, push.



git basics

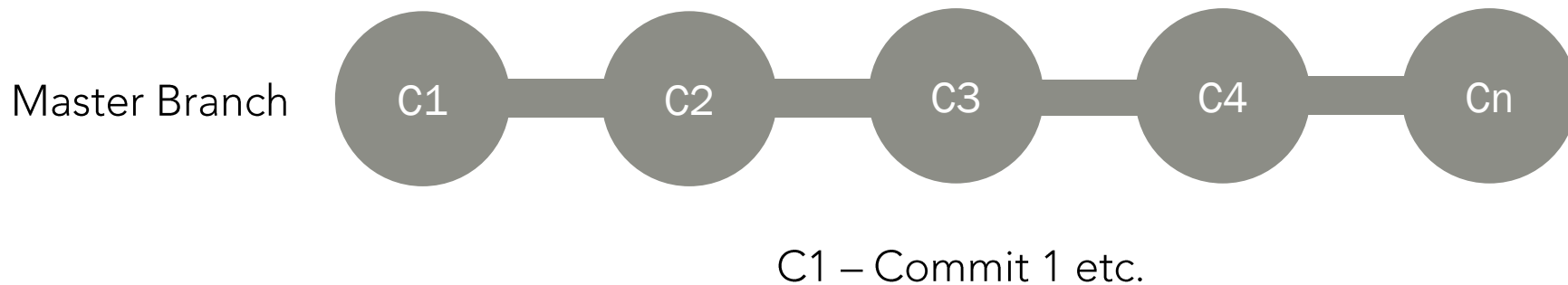
git add; commit; revert; status; push; pull; merge; checkout;

Git Basics - Definitions

- Repository
 - A container for the project that you want to use (track) with Git. There are two types, local and remote (GitHub)
- Working tree
 - The structure and files involved on the branch that you're currently modifying
- Branch
 - Part of the working tree but is separate from the master branch
- Commit
 - The process of placing a file in the repository along with meta-data i.e. who made the commit, the commit summary and history.

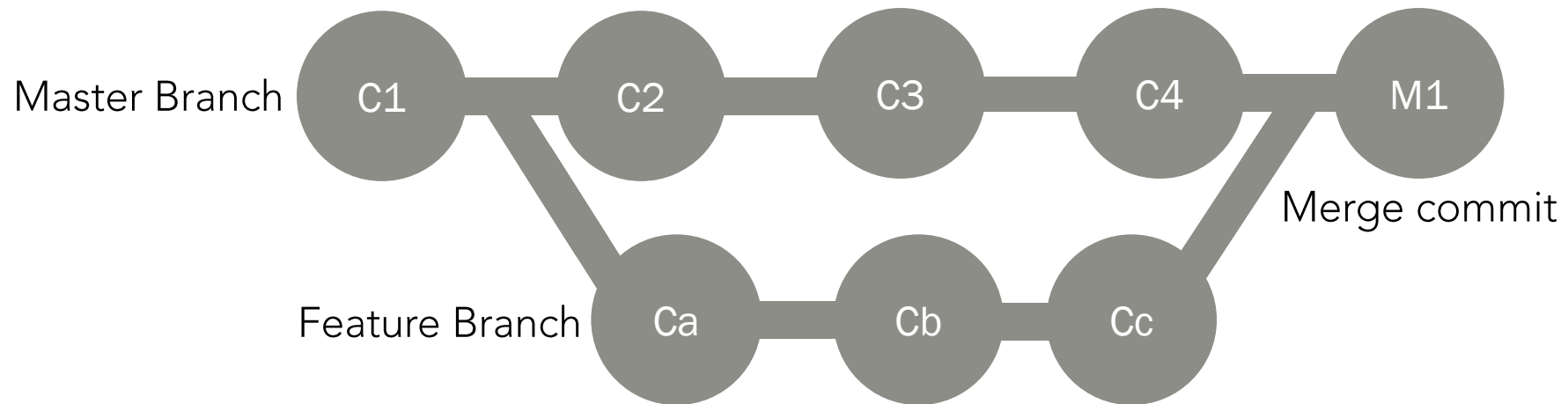
Git Basics – The Working Tree

- The layout of version control is a tree that has one main branch and additional branches
- The **master branch** is where the production ready and tested codebase lives
- Features are developed on additional branches and **merged** into the master eventually



Git Basics - Branches

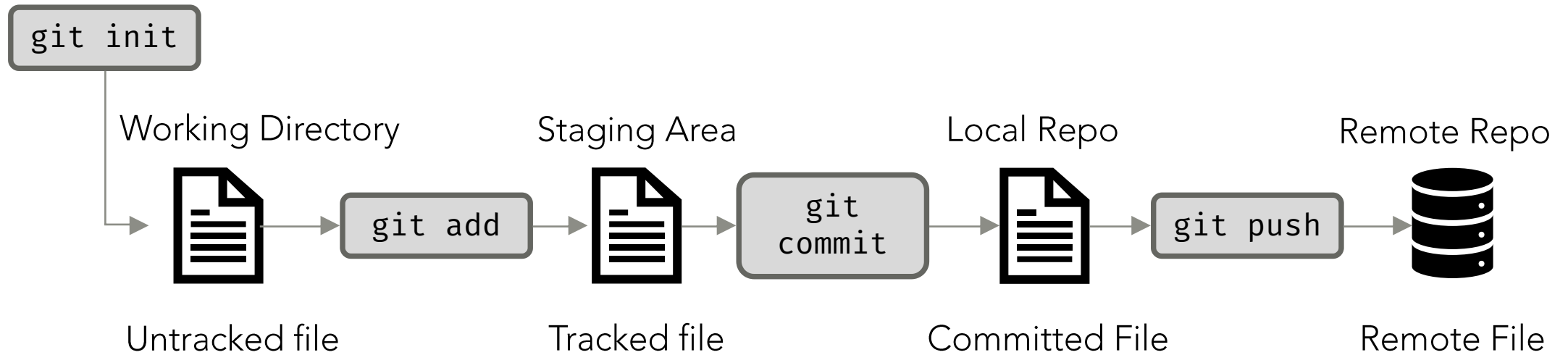
- Branches are used to continue working on the project while not modifying the master branch
- After work on a branch is complete it has to be merged with the master
- Conflicts can arise and require manual intervention to solve



Git Basics - Workflow

- Here's how a workflow in Git usually goes:
 1. Initialize local repository
 2. Add file to staging area to track changes
 3. Commit file to repo (best practice to add a commit message)
 4. Push file to remote repository - GitHub

Git Basics - Workflow




Getting started with Git


- Download Git :
 - Windows and MacOSX <https://git-scm.com/>
 - Windows <https://cmder.net/> - download the version with Git
- You're free to use any command-line tool you wish Cmder, Powershell, Command prompt, Bash
- For Cmder after extraction and running cmder.exe, accept all the warning prompts

Getting started with Git

- Make sure it's installed – Open Terminal
- Windows users – Ignore if using cmdr

-  + X – Open Power Menu
- Select Powershell

- Mac users

-  + space – Open Spotlight
- Type Terminal and open it

Getting started with Git

- Check that git is installed

```
git --version
```

- Output should be

```
git version <some version of git here>
```

Getting started with Git - config

- Configure user email

```
git config --global user.name <yourusername>
```

- Configure username

```
git config --global user.email <your@email.com>
```

* Don't include the angle brackets

* Use the username and email you used when signing up for GitHub

Getting started with Git - init

- Output should be

```
git init
```

```
Initialized empty Git repository in <insert directory here>
```

Getting started with Git - add

- Let's check the status of our repository

```
git status
```

- Create a .txt file. Right click > New > Text Document > add some text > Ctrl +S
- Add files to local repository

```
git add filename.fileextension #Adds a single file
```

```
git add . #Adds all the files in the root directory
```

*Remember to save file first, then run git add

*Lines preceded by '#' are comments and not part of the actual command to be run

Getting started with Git – commit

- Let's check the status of our repository

```
git status
```

- Add files to local repository

```
git commit -m "Your commit message" #adds staged changes to branch
```

Getting started with Git – log

- Check the log for the repository

```
git log
```

- Summarize log information

```
git log --oneline
```

Getting started with Git – revert

- Uh oh, I've made an error in my commits, what to do? 🙄
- Three options:
 1. Checkout commit ✅ - goes back in time, doesn't allow edits to be saved just a preview
 2. Revert commit ⚠️ - 'removes' a single commit from commit history
 3. Reset commit ☠️ – obliterates all commits to a single point in time

Getting started with Git – revert

- Before we continue, add four lines of random text to the .txt file. For each line added, make an add a commit
- Don't forget to save the file before adding and committing
- Write descriptive commit messages
- Experiment with the revert commands

Getting started with Git – revert

- To undo any commits we need the commit ID.

```
git log --oneline #<commit id> <commit message>
```

- Use the commit ID with your applicable command

```
git checkout <commit id>
```

```
git revert <commit id>
```

```
git reset <commit id> --hard #only use --hard when absolutely sure
```

- A interactive prompt will appear at times.
 - Simply press Shift + :
 - Type wq and then hit enter

Getting started with Git – branches

- What if we need to work on a feature. It would be wise to take the existing codebase, work on it but not affect the main files till the feature is ready.
- Branches solve this problem by creating a new path off the master branch that you can freely modify without changing the master branch
- To create a new branch

```
git branch <branchname>
```

- To switch to that branch

```
git checkout <branchname>
```

- To create and automagically switch

```
git checkout -b <branchname>
```


Getting started with Git – branches

- To delete an unmerged branch

```
git branch -D <branchname>
```

- To delete a merged branch

```
git branch -d <branchname>
```

Getting started with Git – branches

- Once work is completed on a branch, it usually has to be merged into the master
- Conflicts can arise during merged and have to be handled manually

Getting started with Git – merge

- After work on a branch is completed, it is usually merged with the master
- To merge a branch switch to the branch that will have the final result of the merge
- E.g. if branch-a had to be merged with master, switch to master and then do the merge
- Switch to appropriate branch

```
git checkout <branchname>
```

- Merge branch with selected branch

```
git merge <branchname>
```



The hub for gits



Getting Started - GitHub

- Now that you're a master in local repositories, it's time to complete your skillset
- We're going to push (export) our local repository (Git) to a remote repository (GitHub)
- To get started:
 - Create a GitHub account: <https://github.com/>
 - Make a new repository, don't initialize with a readme
- Link the remote repo with your local one

```
git remote add origin <link to repository>  
#https://github.com/username/reponame
```

Getting Started - GitHub

- Assuming that you've added and committed files to the repository already, it's time to push the master branch

```
git checkout master  
git push --set-upstream origin master
```

- Wait, what if I want to push a branch that's not the master?

```
git push --set-upstream origin <branch-name>
```

- Before we learned to merge branches with the master once we were done. This is **not advised on multi-developer** projects. In that case, only push your branch and then start a pull request on the web GUI to do the merge

* The --set-upstream switch is only required when switching branches, it can be discarded after it's first use. Sometimes `git push -u origin <branch-name>` is required

Getting Started - GitHub

- A simpler way would be to set up the local and remote repos then clone the remote repo. This avoids a lot of setup overhead and gets you up and running quickly.
- So create a repo on the web, add a .gitignore (search visual studio) and check init with readme
- On your terminal cd into the folder you want to create your project in. Then:

```
git clone <repo_url>  
cd <downloadedrepo>
```

- Then you can work as usual without having to initialize the repository etc.
- What does git pull do?



If only life itself was this simple

Handling non-essential or secret files

- Projects will often contain non-essential files:
 - *Temporary build files,*
 - *machine specific files,*
 - *large assets (fonts, pictures, videos)*
- Projects may have secrets* as well:
 - *API Keys that are setup for test platforms (not secured for production use)*
 - *Encryption keys or passwords*
- How do we avoid committing these files? Manually committing each file is tedious

*If your project requires API keys or passwords, consider using an Environment Variable.

Never use plaintext keys in your code.

Never commit API keys to a public repo (GitGuardian will immediately notify you)

Handling non-essential or secret files

- We have to create a .gitignore file. This instructs Git to pay no attention to any activity in the file names or folder directories listed in .gitignore.
- Some code editors create a .gitignore file for you. If this didn't happen go to: <https://www.gitignore.io/> and search for your IDE - Visual Studio (Code)
- Create new file in IDE and copy over the result from the above search
- Add in your custom directory (doesn't need to exist) in the .gitignore file at the end

```
# My custom directories  
/secrets
```

- Save file

Handling non-essential or secret files

- Tell git to clean out the cache and use the new .gitignore file
- Commit all existing files
 - *Ensure secrets have not been added as yet*
 - *Ensure .gitignore has directory or file to be ignored*

```
git commit -m "Before adding secrets, modding .gitignore"
```

- Clear existing repository cache

```
git rm -r --cached .
```

- Add secret to project

Handling non-essential or secret files

- Add to staging area

```
git add .
```

- Check to ensure git followed the .gitignore file

```
git status
```

- Output

```
On branch master  
nothing to commit, working tree clean
```

- You committed your API key to a public repo, what now?
 - Get a new key
 - Hard reset commit history

Wrapping up

- Set up Git, GitHub and .gitignore before project starts to simplify use during development
- Write descriptive commit messages
- Don't put untested code on the master branch
- Set up your physical environment to be comfortable:
https://www.youtube.com/watch?v=F8_ME4VwTiw
- Interactive cheat sheet (really useful) - <http://www.ndpsoftware.com/git-cheatsheet.html>
- Practice – it'll become second nature and if you have a system crash, you're okay




IN CASE OF FIRE



```
> git add  
> git commit  
> git outnow
```

Questions?

 simeon.ramjit@sta.uwi.edu

 github.com/simeon9696/programmingworkshop