



GRAND CIRCUS

CODING • BOOTCAMPS

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GR  
CIR  
•DE

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

# .NETC# BOOTCAMP

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GR  
CIR  
•DE

GRAND

GRAND

# **GIT & GITHUB**

# GOALS FOR THIS UNIT

1. Intro to the command line
2. Version Control
3. Git & Github
4. Hands-on exercises

The background of the slide is a repeating pattern of the Grand Circus Detroit logo in a light gray color. The logo consists of the words "GRAND CIRCUS" in a serif font, with a stylized building icon above the word "GRAND", and the word "DETROIT" in a smaller font below "CIRCUS".

# INTRO TO THE COMMAND LINE

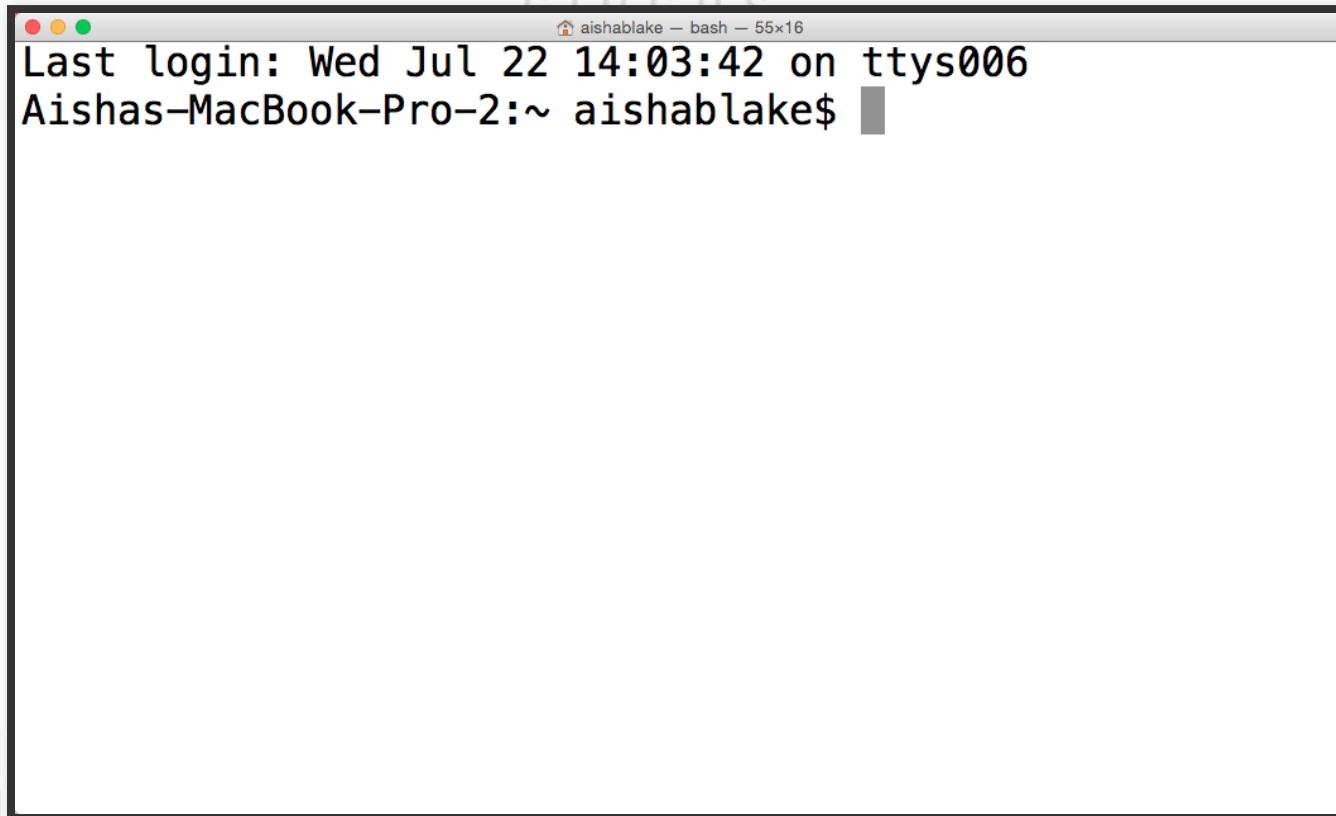
# GETTING STARTED

# INSTALLATION

Download and install **Git Bash**

On a PC, you can use the command line tool that came with your computer, but know that certain commands will be different.

# TERMINAL PROMPT

A screenshot of a macOS terminal window. The title bar at the top shows a home icon, the username 'aishablake', the shell 'bash', and the window size '55x16'. The terminal content shows the last login time as 'Wed Jul 22 14:03:42 on ttys006' and the current prompt as 'Aishas-MacBook-Pro-2:~ aishablake\$' with a black cursor block at the end.

```
aishablake — bash — 55x16
Last login: Wed Jul 22 14:03:42 on ttys006
Aishas-MacBook-Pro-2:~ aishablake$
```

Many command line prompts will end with a dollar sign \$. This is your signal that it's okay to type a new command.

# TERMINAL CHEAT SHEET

There are lots of things we can do with the command line. Once you get used to it, doing things this way can be a lot faster than switching between the mouse and the keyboard, pointing and clicking.

Windows users can use this [cheat sheet](#).



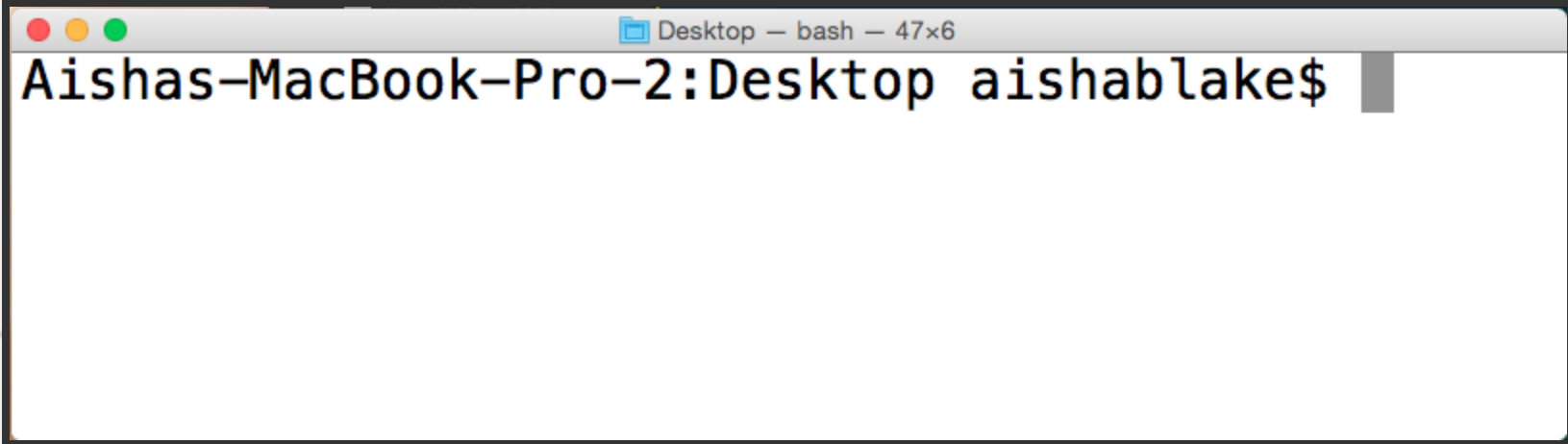
# NAVIGATION

# CHANGE DIRECTORY

The `cd` command allows us to define a path to the directory we wish to navigate to. Executing this command will change where we are in the folder structure. This change should be reflected in our command prompt.

```
$ cd Desktop
```

# GETTING ORIENTED

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, followed by the text "Desktop — bash — 47x6". The main area of the terminal displays the prompt "Aishas-MacBook-Pro-2:Desktop aishablake\$" with a dark gray cursor block at the end of the line.

```
Desktop — bash — 47x6
Aishas-MacBook-Pro-2:Desktop aishablake$
```

The terminal prompt will give us some indication as to where we are within the folder structure.

# CURRENT DIRECTORY

A single period (.) is used to indicate the current directory.

```
$ touch ./example.txt
```

# PARENT DIRECTORY

Two periods (..) indicates the parent directory. This is like moving one level "up" within the folder structure.

```
$ cd ../../another-example.txt
```

# ROOT DIRECTORY

A tilde (~) is used to indicate the root directory. We're essentially saying "go back to the beginning and start from scratch".

```
$ cd ~
```

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

INFORMATION

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

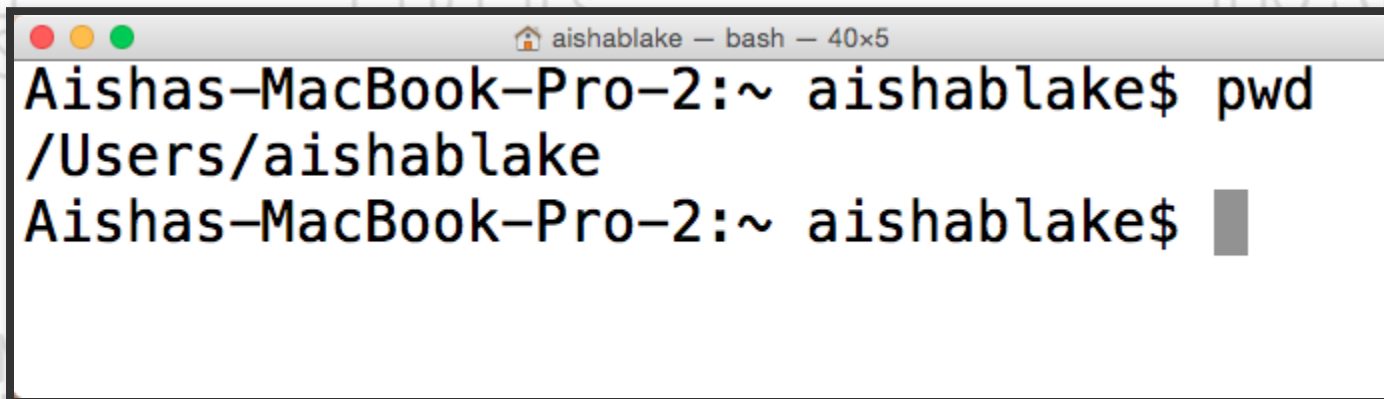
# PRESENT WORKING DIRECTORY

The `pwd` command returns a path showing exactly where we are within the folder structure.

```
$ pwd
```



# YOU ARE HERE

A terminal window with a title bar that reads 'aishablake — bash — 40x5'. The window contains two lines of text: 'Aishas-MacBook-Pro-2:~ aishablake\$ pwd' and '/Users/aishablake'. Below this, the prompt 'Aishas-MacBook-Pro-2:~ aishablake\$' is followed by a dark gray rectangular cursor.

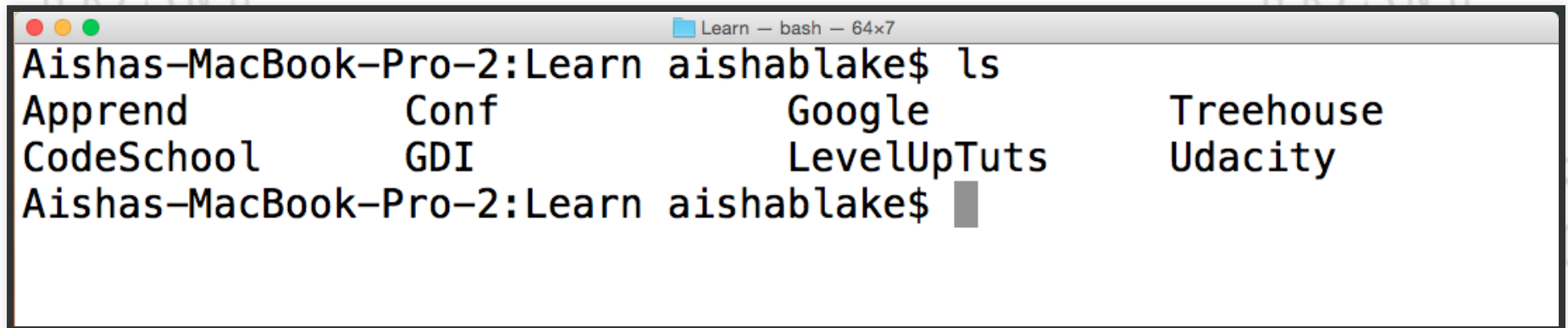
```
Aishas-MacBook-Pro-2:~ aishablake$ pwd
/Users/aishablake
Aishas-MacBook-Pro-2:~ aishablake$
```

# LIST

Use `ls` to see the contents of the current directory.

```
$ ls
```

# TABLE OF CONTENTS

A terminal window titled "Learn — bash — 64x7" with a standard macOS window header (red, yellow, green buttons). The terminal shows the command "ls" being executed in the directory "Aishas-MacBook-Pro-2:Learn aishablake\$". The output lists several items: "Apprend", "Conf", "Google", "Treehouse", "CodeSchool", "GDI", "LevelUpTuts", and "Udacity". The prompt "Aishas-MacBook-Pro-2:Learn aishablake\$" is repeated at the bottom of the terminal output.

```
Aishas-MacBook-Pro-2:Learn aishablake$ ls
Apprend          Conf             Google           Treehouse
CodeSchool       GDI              LevelUpTuts      Udacity
Aishas-MacBook-Pro-2:Learn aishablake$
```

# HELP

When you're not sure what your options are, typing `help` will list all possible commands.

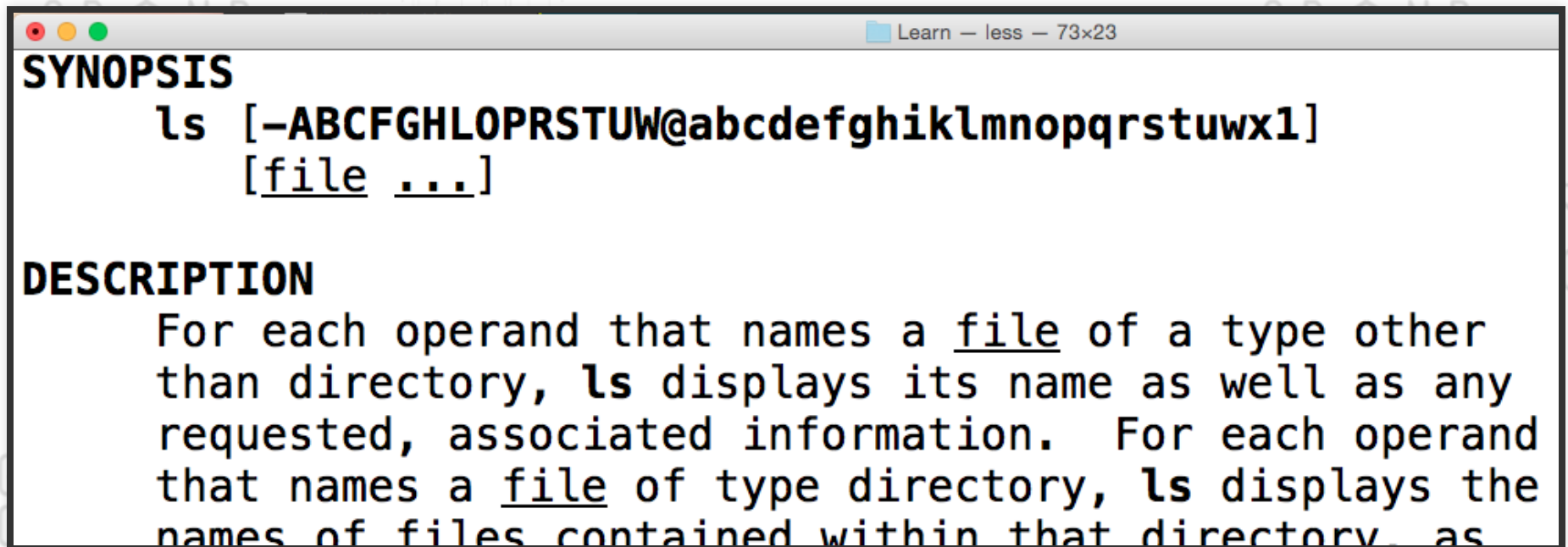
```
$ help
```

# MANUAL

You've figured out what you *can* say, but not what the commands do. Use `man` with any command to see the manual for that command. Hit the 'q' key to escape.

```
$ man ls
```

# NOBODY READS THE MANUAL



A terminal window with a title bar that says "Learn — less — 73x23". The window contains the following text:

```
SYNOPSIS  
  ls [-ABCFGHLOPRSTUW@abcdefghijklmnopqrstuvwxyz1]  
      [file ...]
```

**DESCRIPTION**  
For each operand that names a file of a type other than directory, **ls** displays its name as well as any requested, associated information. For each operand that names a file of type directory, **ls** displays the names of files contained within that directory as

**CREATION**

# NEW FILE

Create a new file by typing the `touch` command followed by the name of the file you wish to create. That file will be added to the current directory.

```
$ touch new-file.txt
```



# NEW FOLDER

Create a new directory (or folder) by typing the `mkdir` command followed by the name of the folder you wish to create. That folder will be added to the current directory.

```
$ mkdir my-stuff
```

# PRO TIPS

# PREVIOUS COMMANDS

We can cycle through any previously executed commands using the up and down arrows. This is useful for a number of reasons:

- Correcting small errors in long or complicated commands
- Retyping frequently used commands
- Recalling the steps that led to the current state

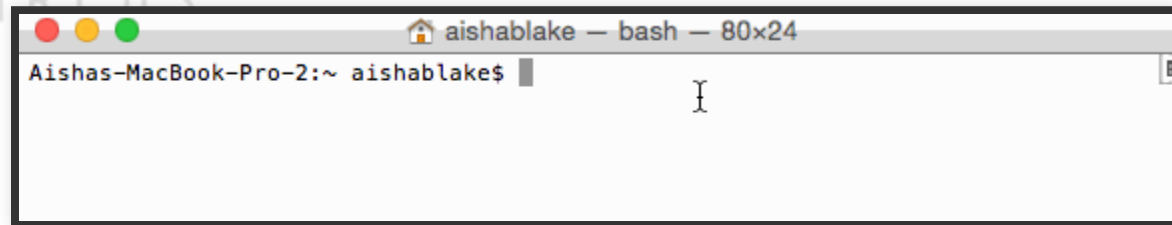
# MULTIPLE COMMANDS

We can string multiple commands together with two ampersands (&&). These commands will be executed in order.

```
$ mkdir my-stuff && cd my-stuff
```

# AUTOCOMPLETE

Pressing `tab` once you've started typing a directory or file name will trigger autocomplete.



# CLEARING THE SCREEN

The `clear` command (you guessed it) *clears* the terminal window. We can also use the keyboard shortcut `Cmd+K`.

# VERSION CONTROL

Version control (sometimes called source control) is a system that manages changes to a program, website, or other collection of files.



Version control facilitates two key processes in development:

- Collaboration
- Managing changes to the codebase

# COLLABORATION

Version control allows you to work on the same project simultaneously with other people and helps to avoid and manage the conflicting changes.

# MANAGING CHANGES

Version control systems allow for seamless integration of new code, easy viewing of previous changes, and makes it easy to undo a change set. It also allows us to revert documents back to a previous state.

## Kinds of version control:

- Centralized
- Distributed

Centralized version control systems are run on one central server infrastructure. Each collaborator checks out the code from and merges changes into the main server.

Distributed version control systems allow each collaborator to maintain a separate repository of the code which can be periodically reconciled with other peer repos.

The background of the image is a repeating pattern of the Grand Circus Detroit logo in a light gray color. The logo consists of the words "GRAND CIRCUS" in a serif font, with a stylized building icon above the word "GRAND", and the word "DETROIT" in a smaller font below "CIRCUS".

# VOCABULARY

# WHAT DOES IT ALL MEAN?!

We've perhaps already thrown around a couple of words that didn't make sense at the time. Well, now is the time for all to become clear!

Note Remember that you can (and should) always just ask if we say something you don't understand!



The background of the slide is a repeating pattern of the Grand Circus Detroit logo in a light gray color. The logo consists of the words "GRAND CIRCUS" in a serif font, with a stylized building icon above the word "GRAND", and the word "DETROIT" in a smaller font below it.

# REPO

A location where data is stored and managed

# UNTRACKED

When we talk about "untracked" files, we mean that Git isn't tracking the changes made to such files. It's aware that they exist, but that's about it.

# TRACKED

Tracked files are monitored by Git. It keeps tabs on every little change made to tracked files.

The background of the slide is a repeating pattern of the Grand Circus Detroit logo in a light gray color. The logo consists of a stylized building icon above the words "GRAND CIRCUS" and "DETROIT" in a smaller font below it.

# **GIT, GITHUB & THE COMMAND LINE**



# **GIT & GITHUB**

Git is an open source, distributed version control system originally developed by Linus Torvalds.

GitHub is a social coding service that offers hosting for software projects that use Git as their source control.

# **GIT & GITHUB**

The combination of the two have become the gold standard in the OSS community and startup scene. It is gaining major ground in the enterprise space as well.

The background of the slide is a repeating pattern of the Grand Circus Detroit logo in a light gray color. The logo consists of the words "GRAND CIRCUS" in a serif font, with "DETROIT" in a smaller font below it, all enclosed within a stylized circular frame.

# GIT

Open your command prompt:

- Git
- Bash



GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

# INSTALLATION

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

GRAND  
CIRCUS  
•DETROIT•

# VERIFY YOUR INSTALL

```
$ git --version
```

If you don't get some kind of error, you're good.

```
git version 2.4.1
```

# IF YOU'RE NOT GOOD

If you've never used Git before, you may need to download it.

Get Git!

While we're at it, if you haven't already...

Create a GitHub account!

# BASIC CONFIGURATION

```
$ git config --global user.name "Your Name"  
# sets the default name for git to use when you make a commit  
  
$ git config --global user.email "you@email.com"  
# sets the default email for git to use when you make a commit
```

**.GITIGNORE**

# WHY WE IGNORE

Sometimes, there will be private files that you want to keep private. Other times, you'll have files that get generated automatically that you don't necessarily want to have to keep track of.

# THE SOLUTION

We can use a special file called `.gitignore` to list out everything we don't want Git to track.

A good item to toss in there right now is `.DS_Store`.

# **GIT COMMANDS**



# 8 GIT COMMANDS YOU NEED

These cover most of what you need to do on a daily basis.

- `git init`
- `git clone`
- `git status`
- `git add`
- `git commit -m`
- `git push`
- `git pull`
- `git reset`

# **GIT INIT**

Creates a new git repository in the current folder  
including all child folders

```
$ cd Desktop // or wherever you want the code to live  
$ mkdir git-demo && cd git-demo  
$ git init
```

# NEW FILES

Create a new file in the new empty repo.

Call it < yourname >.txt

```
$ touch aisha.txt
```

# **GIT STATUS**

Reports the current status of the repo, such as whether any files have been modified or new files have been created.

```
$ git status
```

# **GIT ADD**

Adds untracked files to the repo AND adds a tracked file's changes to the staging area, making it ready to be committed. `git add` must be called with a parameter that is the path to the file(s) you wish to add.

```
$ git add aisha.txt  
$ git status
```

# STAGING AREA

There are 4 states a file can exist in a repo.

- Untracked
- Tracked
- Changed
- Staged
- Committed (which is really just back to 'Tracked')

# GIT ADD

- After you add the file, check your status again. You should see that the file is now being tracked and is ready to be committed.
- Change the file in your editor in some way and save it. Run another `git status`. What happened?
- `git add` the file again. Check your `git status` again. What happened?

# **GIT COMMIT -M < MESSAGE >**

- Once you have code staged, you can commit the change to your repo. This will create an anchor point that you can build from or return to if needed.

Protip: Don't forget the `-m`. If you do, you will be thrown into an in-terminal editor and that will ruin your whole day.



# GIT COMMIT -M

Commit the new file to your git repo.

```
$ git status  
$ git commit -m "I added a new file!"
```

I check my status like a crazy person. Usually to make sure I'm not committing something I don't want to. `git status` will be one of your most used commands.

# BONUS!

`git log` lets you view your repo's commit history

```
$ git log
```

(you may need to press 'q' to exit the log)

There are tons of options you can add to the `git log` command to customize how it looks.

**GITHUB**

The background of the slide is a repeating pattern of a light gray watermark logo. The logo consists of a stylized house icon with a flag on top, followed by the text "GRAND CIRCUS" and "DETROIT" in a smaller font below it.

# GITHUB

In order to push our code to GitHub, we will need to create a remote repository for it to live.

Code with me!

# PUSHING TO A REMOTE

Once your changes are committed locally and you have a remote repository you can push to, you can do a `git push` to push your code to that repo.

# PUSHING TO A REMOTE

`git push` pushes a local repo to a remote location.

```
$ git push origin master
```

```
#or
```

```
$ git push
```

# PULLING FROM A REMOTE

`git pull` pulls changes from a remote repo and attempts to merge them with your local changes. You must be in sync with the latest changes to a remote before you can push changes up to it.

```
$ git pull origin master
```

```
#or
```

```
$ git pull
```

# CLONING A REPO

`git clone` creates a local copy of a remote repository in your local file system allowing you to make local changes. Note: This will make a new directory of the repo's root folder whenever you run this command.

(do this in another directory, not your current project.)

```
$ git clone https://github.com/kroysemaj/git-demo
```



# UNDOING BAD THINGS

`git checkout` allows you to remove the changes to a file before it has been staged.

Make some changes to your js file and save them.

```
$ git checkout aisha.txt  
$ git status
```

All of your changes will be reset to the state of the file in the last commit.

# UNDOING BAD THINGS

With `git reset` you can undo changes you've made or completely destroy all of your changes if you make a real mess of things.

Make more changes to your js file. Now stage them.

```
$ git status  
$ git reset HEAD aisha.txt  
$ git status
```

This will discard all of the staged changes to your file.

file.

## WRAP UP

Git is awesome but it's a little tough to grok at first. You will be learning it in the best way, you're being forced to. We will use git for all of our project work. I encourage you to do the same for your personal projects as well.

Other resources:

- Try Git
- git immersion

# **GETTING STARTED**

Put your last lab up on GitHub!

All labs from this point forward will be tracked using Git and hosted on GitHub. You'll turn those labs in by submitting a link to the corresponding repo.

# INSTRUCTIONS

1. Create a new repo on GitHub.
2. Initialize a new Git repo in your personal site's root directory.
3. Add everything and make your initial commit.
4. Add and push to your remote repo.