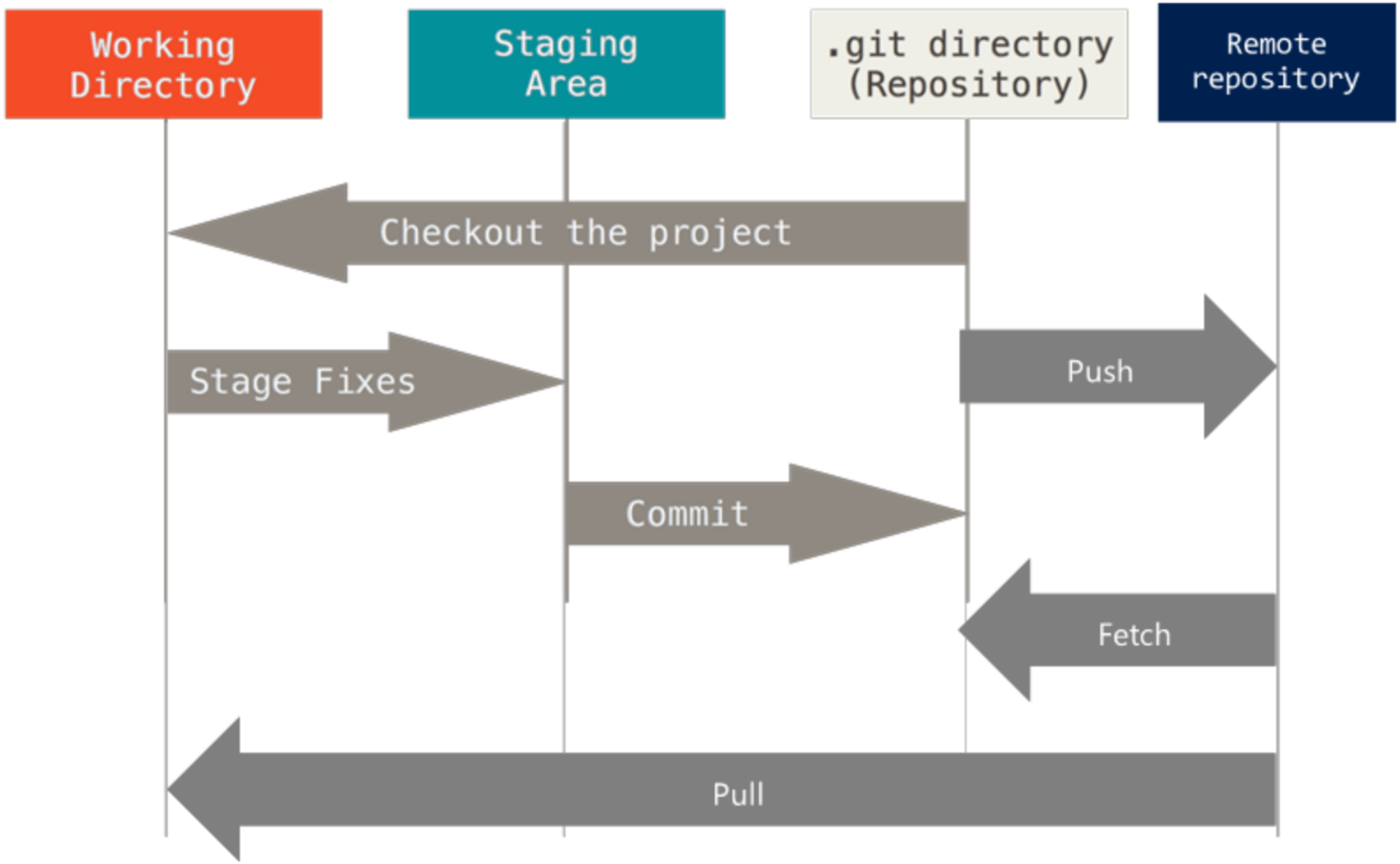
Git Tutorial

Course 2

By Waseem AlBizreh

Git Basics


Working Directory
Staging Area
.git directory (Repository)
Remote repository
Checkout the project
Stage Fixes
Push
Commit
Fetch
Pull

Git Basics

Initializing a Repository:

- If you have a project directory that is currently not under version control and you want to start controlling it with Git. Just type:

```
git init
```

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton.

- You can start version controlling existing files with a few git add commands that specify the files you want to track, followed by a git commit:

```
(use "git add <file>..." to update what will be committed)
```

to update all files that will be committed : `git add *`

Git Basics

Cloning an Existing Repository:

- If you want to get a copy of an existing Git repository, the command you need is git clone

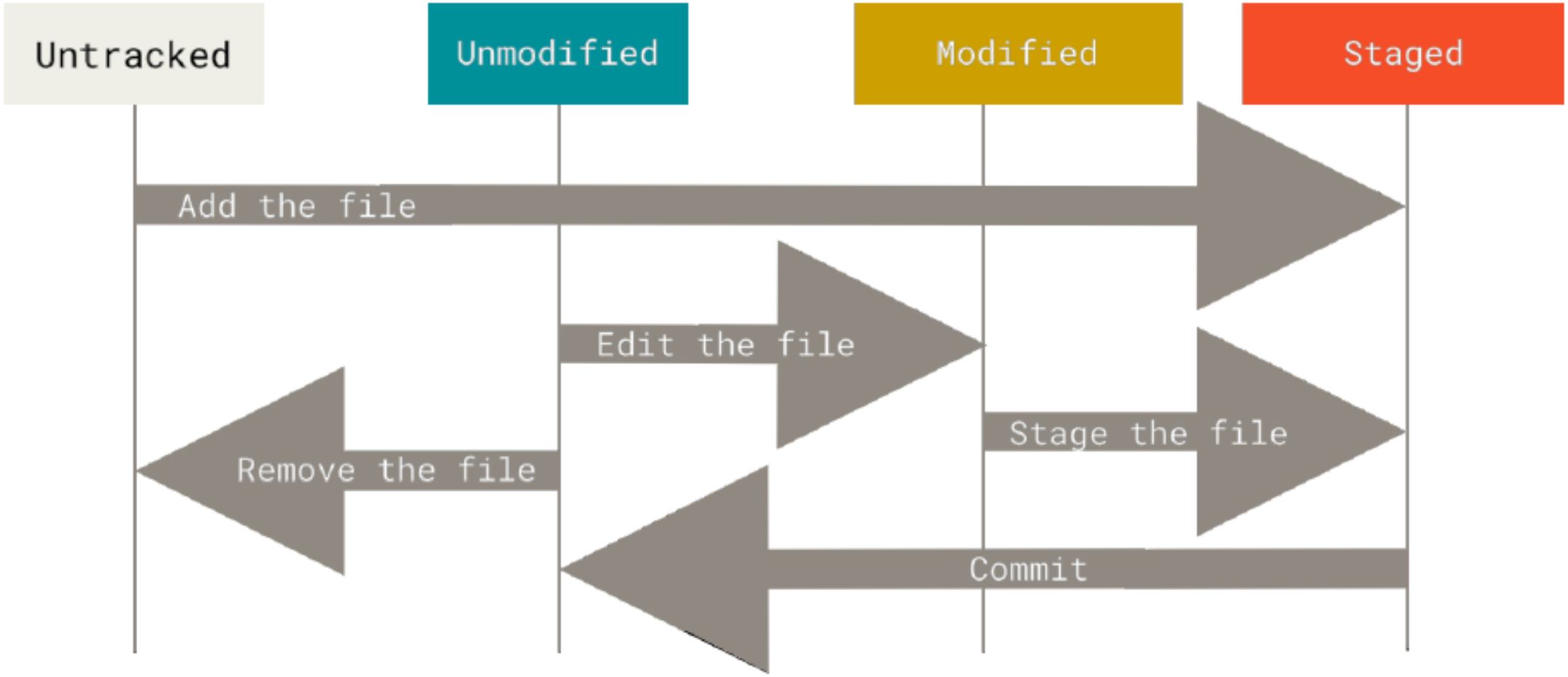
  You clone a repository with git clone <url>.

  For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

```
git clone https://github.com/libgit2/libgit2
```

Git Basics

Recording Changes to the Repository



Git Basics

Recording Changes to the Repository

Each file in your working directory can be in one of two states: tracked or untracked.

- **Tracked** files are files that Git knows about (unmodified, modified, or staged).
- **Untracked** files are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area

Note: When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

Git Basics

Recording Changes to the Repository

- **Checking the Status of Your Files**

  The main tool you use to determine which files are in which state is the git status command:

```
git status
```

- **Short Status**

  Git has short status flag so you can see your changes in a more compact way.

  run git status -s or git status --short

```
git status -s
```

Git Basics

Recording Changes to the Repository

- **Viewing Your Staged and Unstaged Changes**

  git diff shows you the exact lines added and removed.

  To see what you've changed but not yet staged, type git diff with no other arguments:

```
git diff
```

- **Short Status**

  If you want to see what you've staged that will go into your next commit, you can use git diff --staged. This command compares your staged changes to your last commit:

```
git diff --staged
```

Git Basics

Recording Changes to the Repository

- **Committing Your Changes**

  The simplest way to commit is to type git commit:

```
git commit
```

Note: Remember that anything that is still Unstaged won't go into current commit.

- you can type your commit message inline with the commit command by specifying it after a -m flag, git commit –m "<message>"

```
git commit -m
```

Git Basics

Recording Changes to the Repository

- **Unmodifying a Modified File**

  git checkout -- <file>… to discard changes in working directory

  Any local changes you made to that file are gone — Git just replaced that file with the last staged or committed version

```
git checkout -- <file>...
```

- **Unmodifying with git restore**

  Git version 2.23.0 introduced a new command: git restore.

  It's basically an alternative to git checkout which we just covered.

```
git restore <file>
```

Git Basics

Recording Changes to the Repository

- **Unstaging a Staged File**

  use git reset HEAD <file>… to unstage

```
git reset HEAD <file>...
```

- **Unstaging with git restore**

  Git version 2.23.0 introduced a new command: git restore. It's basically an alternative to git reset which we just covered.

  use git restore --staged <file>... to unstage

```
git restore --staged <file>
```

Git Basics

Recording Changes to the Repository

- **Viewing the Commit History**

  The most basic and powerful tool to show commit history is the git log command.

```
git log
```

  One of the more helpful options is -p or --patch, which shows the difference (the patch output) introduced in each commit.

  Note: You can also limit the number of log entries displayed, such as using -2 to show only the last two entries.

```
git log -p -2
```

Git Basics

Working with Remotes

- To be able to collaborate on any Git project, you need to know how to manage your remote repositories.

- Collaborating with others involves managing these remote repositories and pushing and pulling data to and from them when you need to share work

- **Showing Your Remotes**

  To see which remote servers you have configured, you can run the git remote command.

  You can also specify -v, which shows you the URLs that Git has stored for the shortname to be used when reading and writing to that remote:

```
git remote -v
```

Git Basics

Working with Remotes

- **Adding Remote Repositories**

  To add a new remote Git repository as a shortname you can reference easily, run

  git remote add <shortname> <url>:

```
git remote add pb https://github.com/paulboone/ticgit
```

Note: Now you can use the string pb on the command line in lieu of the whole URL.

Git Basics

Working with Remotes

- **Fetching and Pulling from Your Remotes**

  to get data from your remote projects, you can run git fetch <remote>:

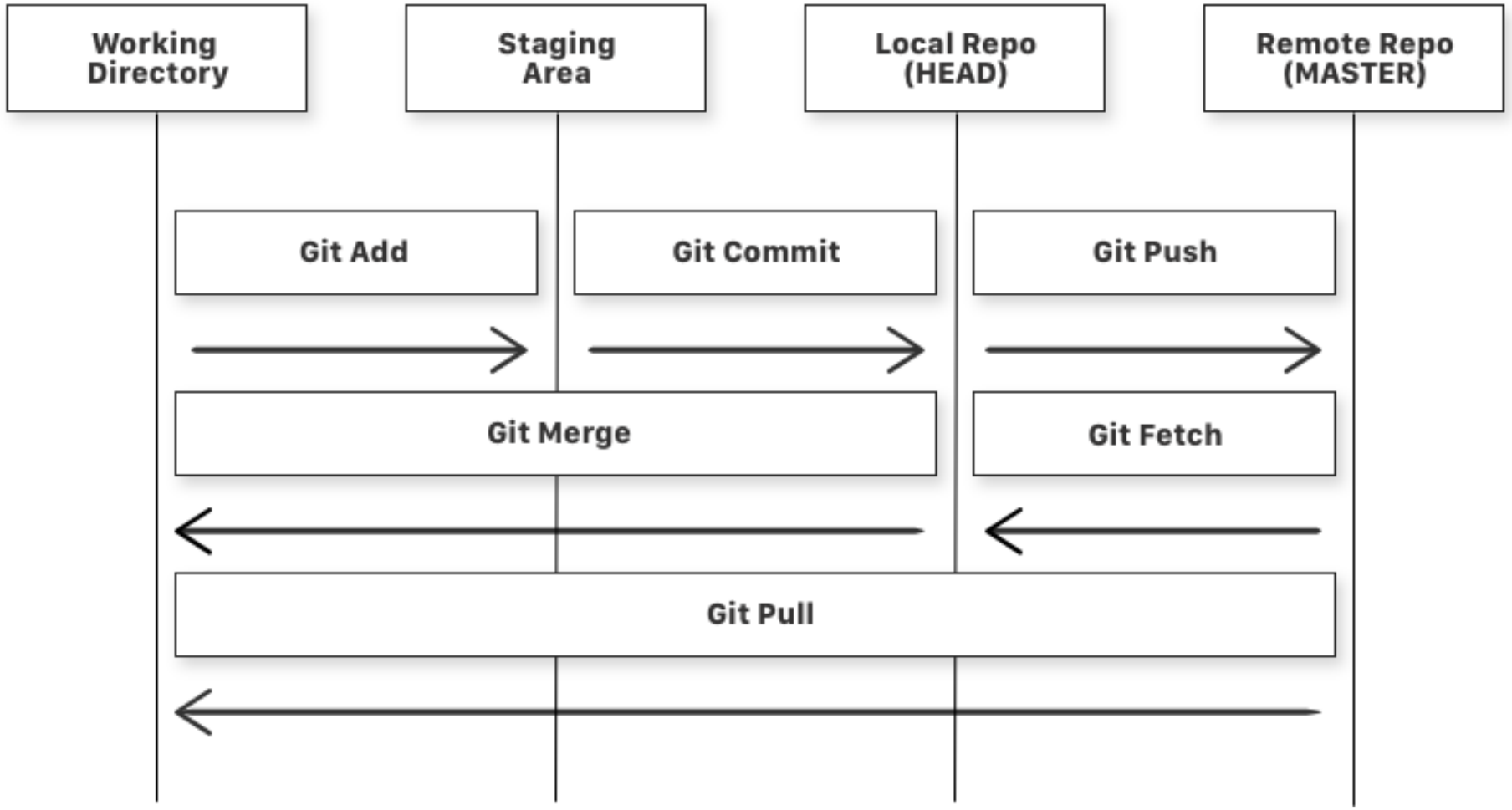
```
git fetch <remote>
```

  The command goes out to that remote project and pulls down all the data from that remote project that you don't have yet.

  Note: It's important to note that the git fetch command only downloads the data to your local repository it - doesn't automatically merge it with any of your work or modify what you're currently working on.

Git Basics

Working with Remotes



Git Basics

Working with Remotes

- **Fetching and Pulling from Your Remotes**

  you can use the git pull command to automatically fetch and then merge that remote branch into your current branch.

```
git pull
```

  Running git pull <remote> <branch> generally fetches data from the server you originally cloned from and automatically tries to merge it into the code you're currently working on.

Git Basics

Working with Remotes

- **Pushing to Your Remotes**

  When you have your project at a point that you want to share, you have to push it upstream. The command for this is simple: git push <remote> <branch>.

  ex: If you want to push your master branch to your origin server

```
git push origin master
```

- **Renaming and Removing Remotes**

  You can run git remote rename to change a remote's shortname. For instance, if you want to rename pb to paul, you can do so with git remote rename:

```
git remote rename pb paul
```

Git Basics

Git Aliases

Feature that can make your Git experience simpler, easier, and more familiar: **aliases**

you can easily set up an alias for each command using git config. Here are a couple of examples you may want to set up:

```
git config --global alias.ci commit
git config --global alias.st status
```

It’s also common to add a last command, like this:

```
git config --global alias.last 'log -1 HEAD'
```

Git Basics

Ignoring Files

Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked. In such cases, you can create a file listing patterns to match them named .gitignore.

The rules for the patterns you can put in the .gitignore file are as follows:

- Blank lines or lines starting with # are ignored.
- You can start patterns with a forward slash (/) to avoid recursively.
- You can end patterns with a forward slash (/) to specify a directory.
- You can negate a pattern by starting it with an exclamation point (!).

Note: .gitignore file examples for dozens of projects and languages:

https://github.com/github/gitignore

Git Basics

Ignoring Files

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```


Thank you