Git & Github

Resources

- Learning Git and GitHub on LinkedIn Learning (by Ray Villalobos)
- Git Essential Training: The Basics on LinkedIn Learning (by Kevin Skoglund)
- Pro Git (https://www.git-scm.com/book/en/v2)
- Git for Humans (https://learning.oreilly.com/library/view/git-for-humans/9781492017875/)
- Happy Git with R *(R-flavored)* (https://happygitwithr.com)
- Ten simple rules for taking advantage of Git and GitHub (https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004947)

What is Git?

- Git is a version control system developed in 2005 by Linus Torvalds (creator of Linux) for the development of Linux
  - (Follow up question: What is version control?)
- One of the most widely used modern version control systems
- Manages the collection of files that make up a project

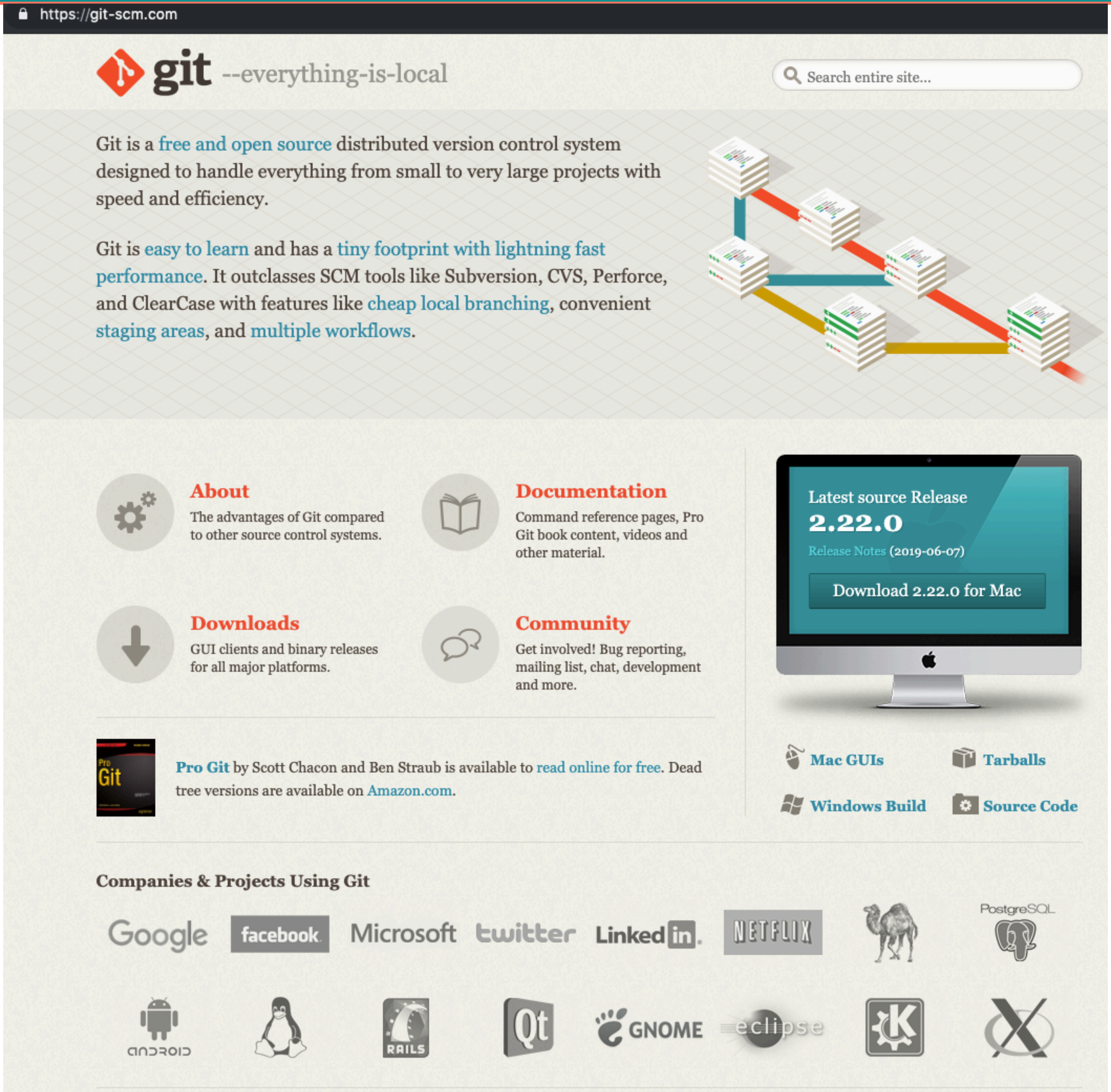
What is GitHub?

GitHub

- GitHub is a web based Git repository hosting service
- In addition to the distributed version control functionality of Git, it adds some of its own features
- GitHub was developed in 2008 and it was sold to Microsoft in 2018 for $7.5 billion

Install Git

git-scm.com



https://git-scm.com

git --everything-is-local

Search entire site...

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Pro Git by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Latest source Release
2.22.0
Release Notes (2019-06-07)
Download 2.22.0 for Mac

Mac GUIs
Tarballs
Windows Build
Source Code

Companies & Projects Using Git

Starting with Git

Configure Git

```
# REQUIRED
> git config --global user.name “My Name”
> git config --global user.email “myemail@usa.com”

## OPTIONAL
# if you want to have the terminal display color
# it might already have this by default
> git config --global color.ui auto
```

Note about the default editor vi

- If you forget to add a commit message (and type **`git commit`** without the **`-m`**), the default editor will pop up for you to add a commit message.

- The default editor is "vi"

  - hit the "i" key (to go into interactive mode)

  - type the commit message

  - to exit: hit "esc" then type **`:wq`**

- You can configure to be a different default editor. See: https://docs.github.com/en/get-started/getting-started-with-git/associating-text-editors-with-git

Creating a Git repository for an existing directory

- Initializing - start a git repo on your local machine

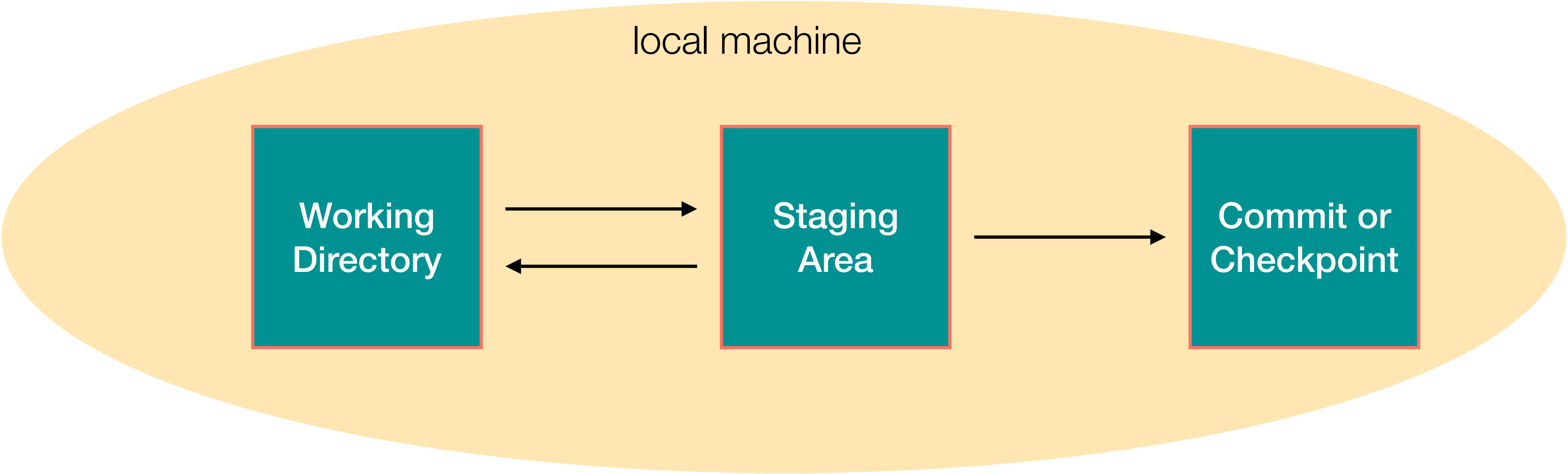
```
# Initialize
# Navigate to desired directory

> git init
```

Let's practice

- Open a terminal (mac) or Git Bash (windows)
- Set up the config file with your name and email if you haven't already
- Download and unzip the MyProject.zip file from LearningSuite
- In the command line navigate to the MyProject folder
- Initialize the project to be a Git Repository
- Verify the .git folder is there
- Check the status by typing `git status`

Git Environments

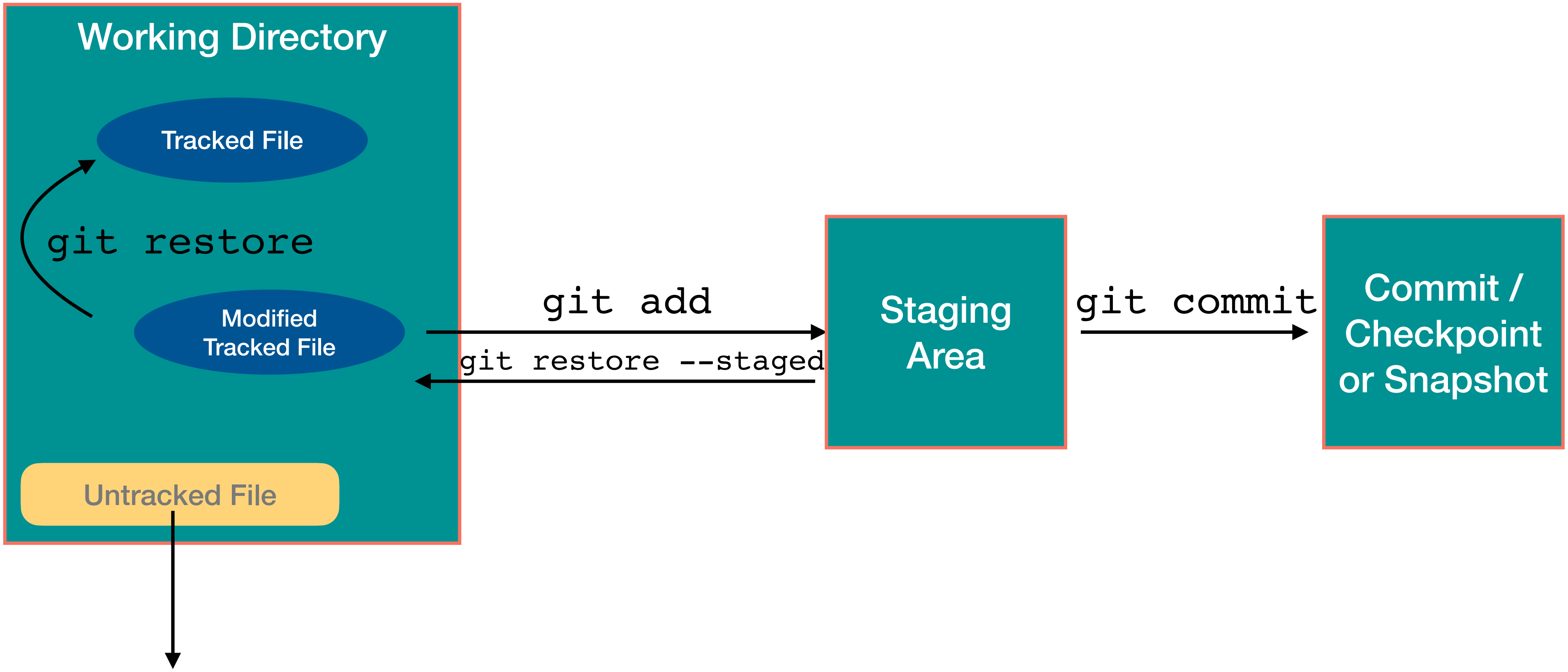
Git Environments



FILES STATES

- Tracked
  - Unmodified (version in WD same as latest commit)
  - Modified (version in WD changed from latest commit)
  - Staged (modified version in staging area)
- Untracked

Exploring Environments



Git doesn't monitor untracked files
New files must be staged then committed to become tracked

Actual git commands

- `git status`
  reports the difference between working directory, staging area, and repository

- `git add [filename(s)]` or `git add .`
  adds files from the working directory into the staging area.
  Note that "`git add .`" will add all files in the current directory into the staging area

- `git commit -m “commit message here”`
  commits files in staging area to the repository. The commit message should describe what the commit is doing in present tense.

- `git restore [filename]` or `git restore .`
  converts files (or directories) that have been modified to the current repository version

- `git restore --staged [file name]`
  unstage a file

- `git log`
  shows all commits in history (in “less” mode—hit q to quit if necessary)

Some common git log options

```
> git log --oneline
> git log --oneline -2

> git log --after="2019-09-04"
> git log --since="2019-09-04"

> git log --before="2 weeks ago"
> git log --until="2 weeks ago"

> git log --author="Shannon"
```

Let's practice

- Stage all the files in the "MyProject" folder
- Make your first commit
- Make a change
- Add and commit again
- Practice with checking the status and log

Ignoring Files

.gitignore

- Sometimes there are file in the repository that we don’t want tracked
- We can ignore files by creating a .gitignore file
- The .gitignore should be in the main folder of the repository
- It can contain names of specific files, specific folders, or patterns

Example .gitignore file

```
dumb.txt   # will ignore the file called "dumb.txt"

*.php #will ignore all files with extension .php

!specific.php  #except for "specific.php"

(the ! means not as in "do not ignore")

folder/subfolder/ # use trailing slash to ignore all files in subfolder
```

My typical .gitignore file

```
.DS_Store
*.ipynb_checkpoints/
```

Helpful Links

- https://help.github.com/articles/ignoring-files
- https://github.com/github/gitignore

Let's practice

- Make a .gitignore file
- Stage and commit the .gitignore file

Branches and Merging

Branching

- Branches allow us to create new versions of the project without changing the "main" project
  - Experiment with adding features
  - Team work (your part is done on a branch)

Git Branch Commands

-
  ```
  git branch
  ```
  list branches ("*" will appear next to the active branch)

-
  ```
  git branch [branch-name]
  ```
  create a new branch

-
  ```
  git checkout [branch-name]
  ```
  or
  ```
  git switch [branch-name]
  ```
  switch to another branch

-
  ```
  git checkout -b [new-branch-name]
  ```
  or
  ```
  git switch -c [new-branch-name]
  ```
  create a new branch and switch to it at the same time

Merging and Deleting

- (navigate to the main branch)

  ```
  git merge [branch-name]
  ```

  merge the specified branch into the current branch

-
  ```
  git branch -d [branch-name]
  ```

  delete a branch if there are no conflicts

  or

  ```
  git branch -D [branch-name]
  ```

  delete a branch forcing git to ignore any conflicts

Typical Git Flow

- Create new feature / fix a problem on a new branch
- Make changes
- Merge back into main
- Delete branch

About Merges

- Two main types of merges:

1. Fast-forward
   - No other changes have been made to master (or current branch), so master is just “fast forwarded” to the point of the merged branch
   - A commit message is not needed
2. Merge commit
   - Changes have been made to both branches and both the changes will be merged.
   - A commit message is recommended
   - Potential for a merge conflict

Merge Conflicts

- Merge conflicts happen when you merge branches that have competing commits, and Git needs a human's help to decide which change to incorporate in the final merge

- See https://help.github.com/en/articles/about-merge-conflicts

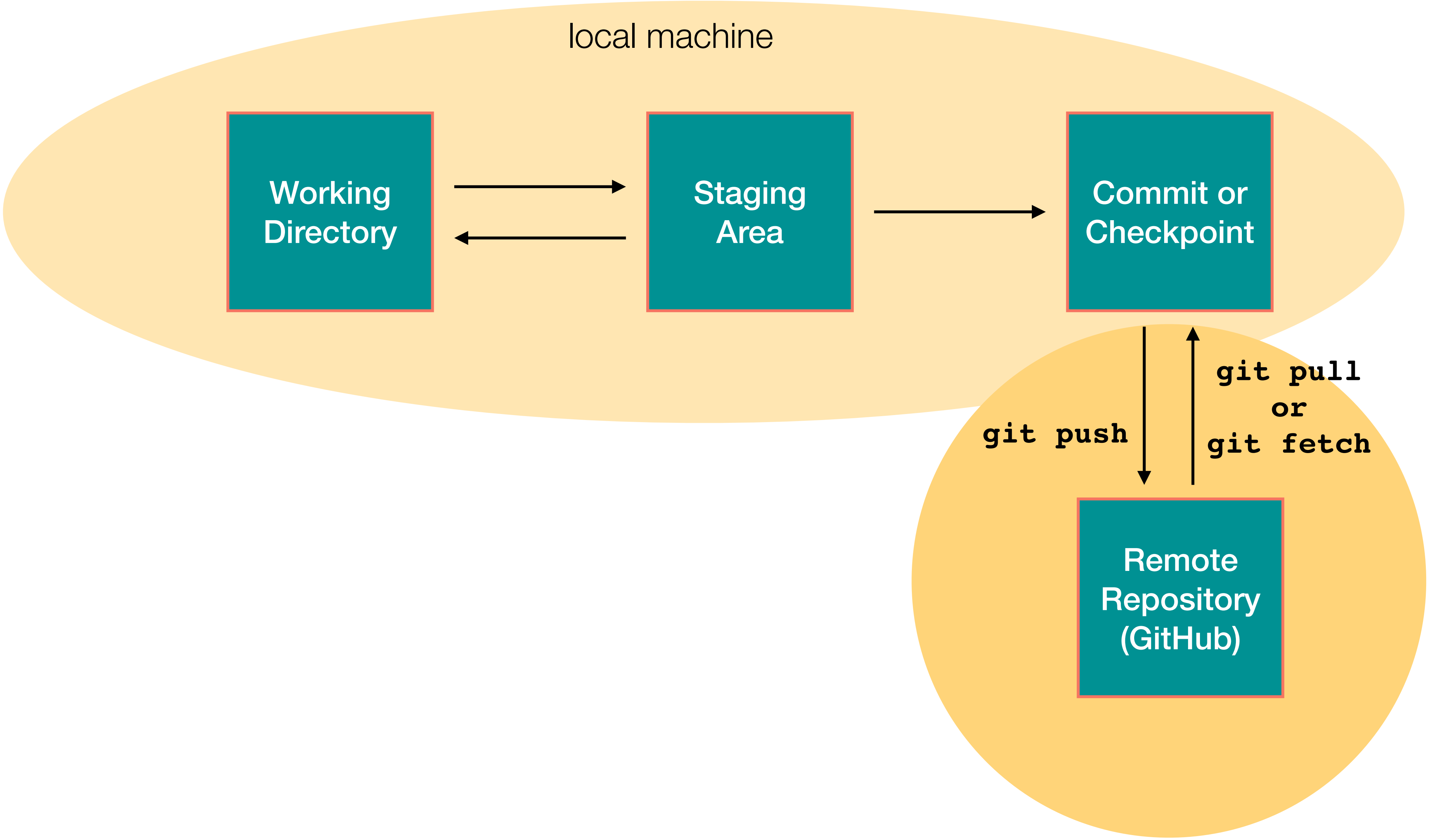
- Merge conflicts are most common when collaborating with others

Reduce Merge Conflicts

- Keep code lines short
- Keep commits small and focused
- Beware of stray edits to whitespace
- Merge often (if possible)
- Sync remote and local work **whenever** a change is made
- *Pull remote* repo into local before starting work
- *Fetch* remote repo and examine changes before *pushing*

remote repositories

Everything we've done so far has been local



Remote repositories

- GitHub is just one cloud based option for remote repositories
- Many companies will have an in-house remote repository location
- Bitbucket
- Google Cloud Source Repositories

Transfer commits from local to remote with "push"

- Pushing refers to sending your committed changes to a remote repository

- When you change something locally, you then **push** those changes to the remote repository, such as GitHub (so others can potentially see them)

Linking a GitHub repo with your local repository

1. Create a blank repository in your GitHub account and copy the URL.

2. On your local machine, add the remote location:

   ```
   git remote add <alias> <url>
   ```

   for example:

   ```
   git remote add origin https://github.com/user_name/repo_name.git
   ```

3. Push the local repo to GitHub

   ```
   git push -u origin main # set-upstream (first time only)
   git push
   ```

   or

   ```
   git push --all # push all the branches
   ```

4. Refresh GitHub to see that it worked

Branch "origin/main"

- When we add the remote (with the alias origin), we are creating a remote branch called "origin/main".

- origin/main works like any other branch except that it can't be checked out

- You can see it with

  - `> git branch -r` (to see remote branches) or

  - `> git branch -a` (to see all branches)

Get changes from the remote to the local repo

```
> git fetch origin
```

- A `git fetch` will get the current version of the remote repo and put it into origin/master
- A `git fetch` is safe because it doesn't change anything on the local repository
- You can see difference between the remote and local by checking out the log

```
> git log origin/main
```

- In order to put any changes into the local repo, you have to do a merge

```
> git merge origin/main
```

Get changes from the remote to the local repo

```
> git pull origin
```

- `git pull = git fetch + get merge`
- A `git pull` is faster, but might overwrite any changes that you've made on the local repo

When working with others

- Always fetch/pull before you start work on your local machine to ensure you have current version of repo

- It is good practice to fetch before you push to see any changes that others have made

Cloning Repository and Changing the Remote

Copy a remote repository

```
> git clone <url>

# git clone will automatically add the remote path

# you can "fork" someone else's repository to make
# the current version into a repo of your own.
# That way you can push any changes that you make
# to a remote that you own.

# To change the remote path
> git remote set-url origin <url>
```

Keep the class GitHub Repo Updated in Your GitHub

1. Fork the repository into your account
2. Select the "Fetch upstream" drop-down
3. Review differences and then click "Fetch and merge"

Let's Practice

- Create a new repository in your GitHub account
- Push the local repo we've been working on to GitHub

GUI Clients for Git

Avoiding the command line

- https://git-scm.com/downloads/guis/
- Many IDEs and/or editors manage Git including:
  - Rstudio
  - Atom
  - Visual Studio

Git can be frustrating

Hang in there and keep practicing

https://xkcd.com/1597/

https://explainxkcd.com/wiki/index.php/1597:_Git

See your git log as Star Wars scrolling

http://starlogs.net/