Preamble

http://bit.ly/2yZWVpX

First 10 minutes will be used to ensure

1. `git` installed on your computer.
2. `Github` account setup & give username to the TAs

Please sit in front of room if you haven't yet setup your environment

If you didn't bring a computer, or are unfamilure with terminal commands, make a friend

Portland DSG

Thanks to New Relic for hosting this talk!



Intro to Git

http://bit.ly/2yZWVpX

This talk will not cover, or expect, specific programming languages

1. Introduce collaborative demo
2. We will introduce vocabulary
3. Understand Simplest Workflow
4. Attendants will contribute to collaborative demo
5. Talk about access control and Github

Goals

If you successfully finish workshop, you will

- be able to collaborate on simple projects
- understand basic vocabulary for git
- know how/what to study next

Why do you Care?

- No more emailing document revisions
- Simpler local directory/file structures
- Remote storage
- Stable workflow
- Easily add new collaborators to project

Your Project!

- After I assign you a **pannel** of content,
- add **pannel** contents to the `README.md` file
- belonging to an **existing repository**
- in **alphabetical order**.
- Finally, share your changes

```
A is for Alice who fell down the stairs
B is for Basil assaulted by bears
C is for Clara who wasted away
...
Z is for Zillah who drank to much gin
```

What is Git?

- Source and version control
- Ledger of work
- Collaboration tool
- Workflow management software

competes with : **hg**, **svn**, **cvs**

What is Github?

- Git service provider
- Account management and access control
- Hosting platform
- Ticket tracker / project management tool

competes with : **GitLab**, **bitbucket**, **coding.net**

What to store (Github)?

- source code (language ambiguous)
- markdown / Jupyter / pdf
- small or static images & data-sets

What NOT to store?

- PASSWORDS, access tokens, or private keys
- compiled binaries
- large images & data-sets
- Non-pars-able documents (Word, Photoshop, ...)

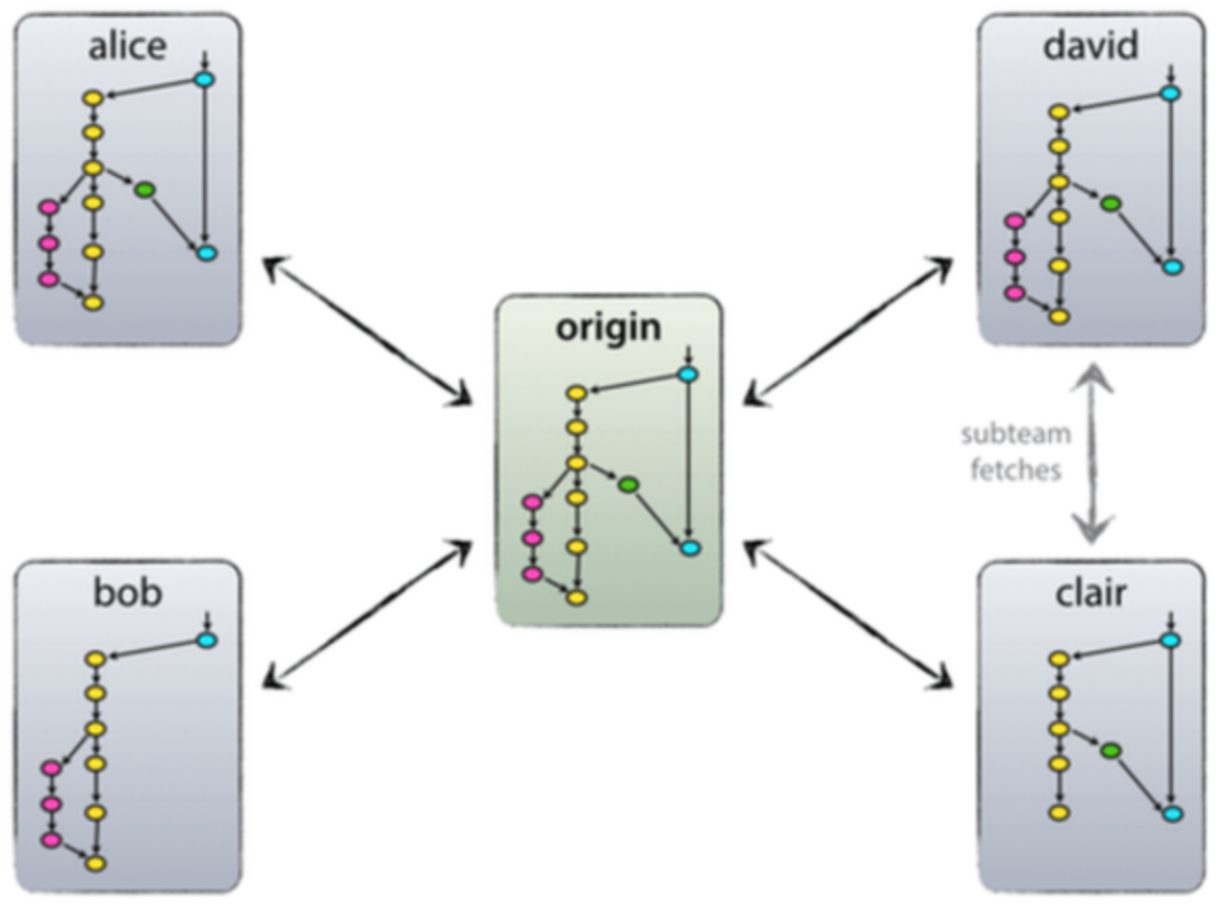
Passwords

```
$ cat secrets.json # this file should not be committed
{
    "password":"MySuperNeatoPassword!#"
}
```

```
import json

# this file should be commited

with open('secrets.json') as fd:
    pwd = json.load(fd)['password']

print(pwd)
```

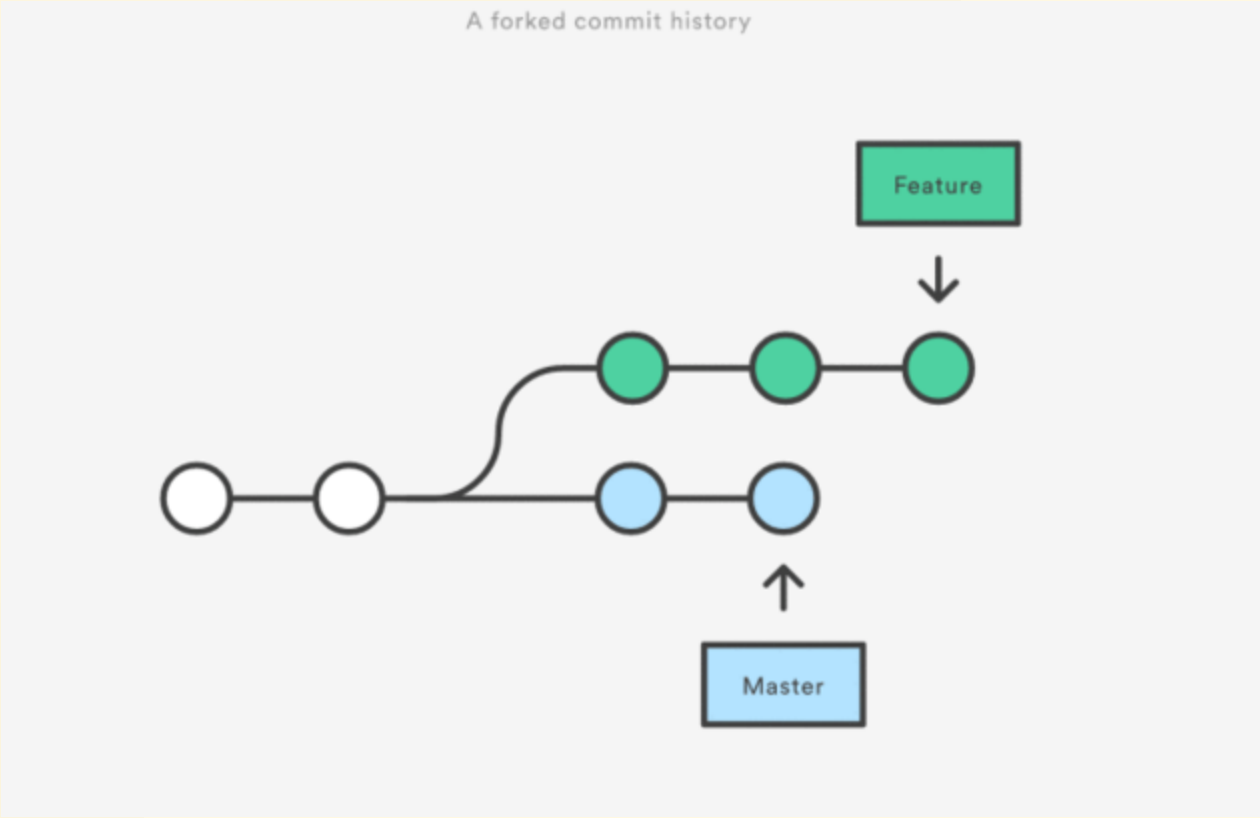
Vocabulary (I)

- `remote` labels alias another location



Vocabulary (II)

- `branch` - encapulates a split in ledger of work
- `checkout` - swaps current working copy to a target



Vocabulary (III)

- `init` / `clone` - Starts Project
- `add`/`commit` and Commit History - Tracks Changes
- `fetch`/`merge` and Merge Conflicts
- `push` - Share Changes
- `diff`, `status`, `log` - What state am I in?

Inspecting the Repository

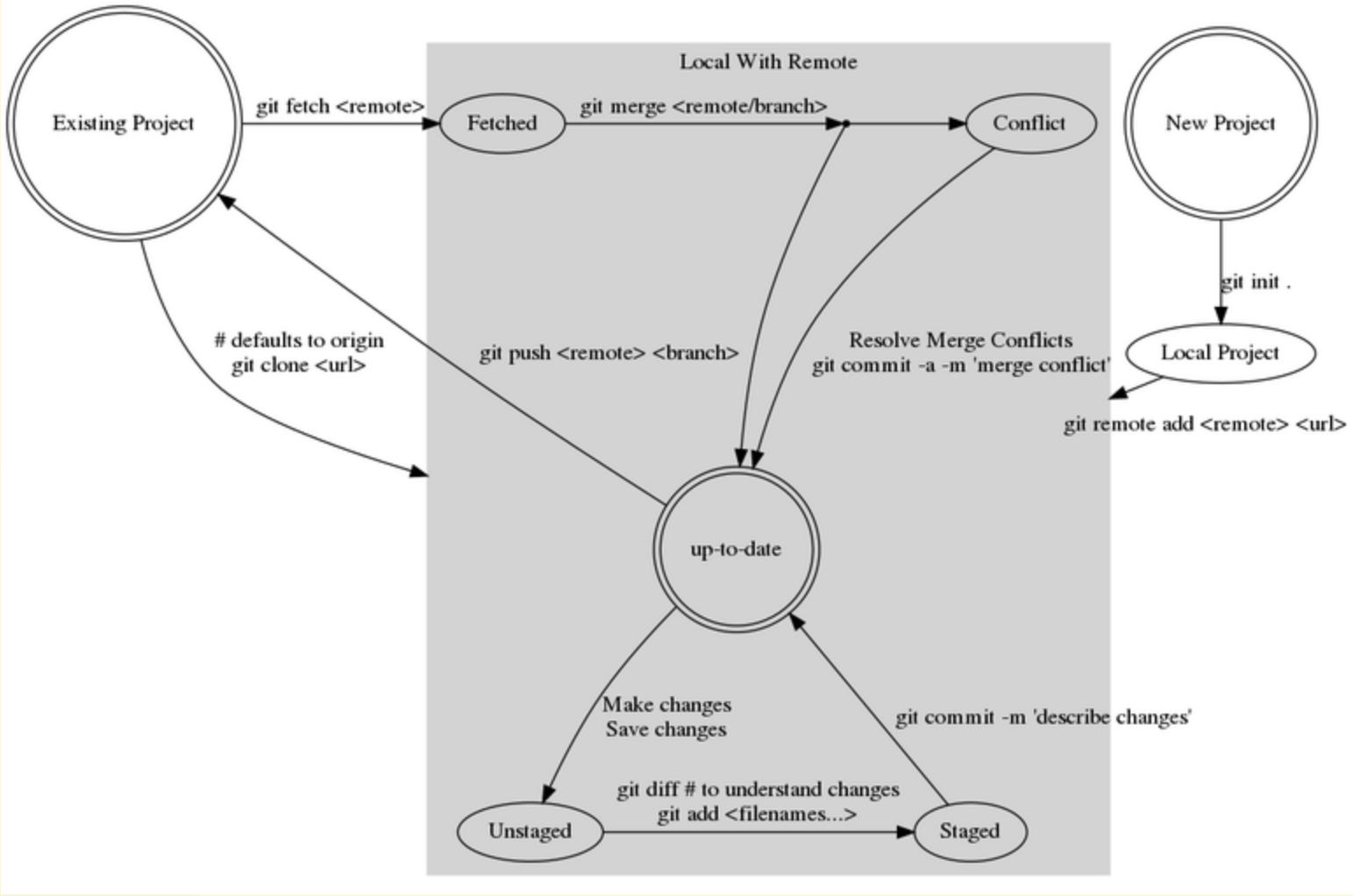
Merge Conflicts

```
$ git status
# On branch branch-b
...
# both modified:
```

```
$ cat styleguide.md
If you have questions, please
<<<<<<<< HEAD
open an issue
========
ask your question in IRC.
>>>>>>>> branch-a
```

```
$ cat styleguide.md
If you have questions, please open an issue or
ask your question in IRC.
```

Workflow Map



Message Flag and Editor

If you don't use the `-m` message flag, you will likely be subject to `vim`. `vim` can be a very frustrating file editor, if you don't bother to learn it.

Look into how to change your default `EDITOR` for your operating system.

Vim

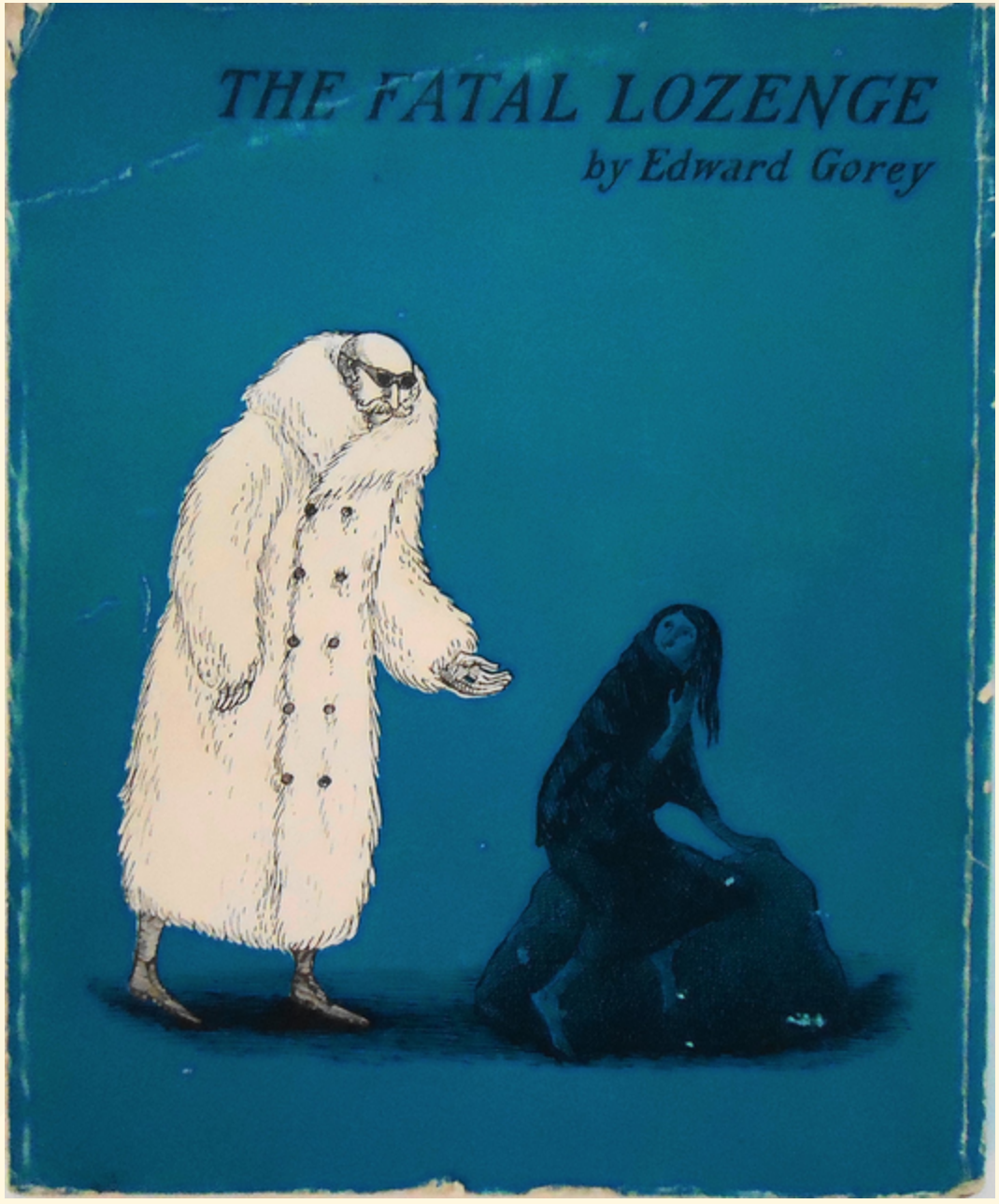
To exit `vim`, Hit the `Esc` key to enter "Command mode". Then you can type `:` to enter "Command-line mode". A colon (`:`) will appear at the bottom of the screen and you can type in one of the following commands. To execute a command, press the `Enter` key.

- `:q` to quit (short for `:quit`)
- `:q!` to quit without saving (short for `:quit!`)
- `:wq` to write and `quit`

Defaults and `config`

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Give me your Username in Exchange for A Ticket


THE FATAL LOZENGE
by Edward Gorey

Rules of the Game (15 min)

- 10 people per repository team {red, blue, black}
- Each pannel represents one entry from poem
- Split pannel text lines with <= 80 characters
- Pannels ordered alphabetically (by second word)
- Your team is done when all pannels added
- Do a `diff` against `origin/master` before a `merge`
- Review the `log` at least once (`q` to quit)

team == red

https://github.com/PortlandDataScienceGroup/red.git

Git Commands

- `status`, `diff`, `log`
- `clone`, `init`
- `fetch`, `merge`, `commit`, `push`, `fetch`

```
$ git clone https://github.com/PortlandDataScienceGroup/ABC
Cloning into 'ABC'...
remote: Counting objects: 500, done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 500 (delta 27), reused 33 (delta 12), pack-re
Receiving objects: 100% (500/500), 6.72 MiB | 1.71 MiB/s, d
Resolving deltas: 100% (286/286), done.
Checking connectivity... done.
$
```

Intermission

Github activities

- Code reviews
- Create / destroy user and organization accounts
- Access control
- Create / destroy repository
- Issue creation / assignment / management
- Gists

Collaborating without Permission

Collaboration within a team is different than from outside, as a consiquence of access control.

- `fork` - Copies repository
- `pull request` - Shares changes back to source

Steps

1. `fork` repository on Github
2. `clone` forked repository to local directory
3. `add upstream` directed toward original repositry
4. Edit files, save, `commit`, then `push` changes forked repository
5. `pull request` against original repository

Branching

Branching allows

- emcapsulation of features
- simple `diff`s between features
- easier `pull requests`

Collaboration Etiquette

- Look for a `CONTRIBUTORS.md` file
- Look for style guides
- Read documentation before collaborating
- Take code review feedback seriously and not personally
- Identify an appropriate `issue` for your skill level
- `rebase -i` to encapsulate solution to single `issue`

To Learn Next

- Github `issues`
- `checkout` - use another version as working copy
- `branch` - encapsulate work
- `rebase`/`rebase -i` - edit branch history
- Learn about branching models
- Learn about version numbers
- `GitFlow` http://nvie.com/posts/a-successful-git-branching-model/

Portland DSG