Intro to Git and GitHub

… if version control is a timeline, let's do some time travelling!

Where We Will Travel

- Overview of Version Control and Git
- Git Basics
- Git Exercise
  - Explore basic Git functions
  - Opportunity to ask question or share knowledge
- Git
  - TortoiseGit
  - PyCharm and VSCode
- GitHub Tutorial (Open Source Contribution Patterns)
  - Forking a project
  - Making a pull request
- Questions? Other Resources… What’s Next?

Your Guide on the Adventure

- Senior Software Engineer
  - at a top Digital Certificate Authority
  - DevOps, certificate lifecycle management, …
- Git background
  - Using Git since mid-2008
  - Advocated using Git as a front-end to Subversion (SVN)
  - Converted multiple CVS and SVN repos to Git
- Data Engineering interests
  - Machine Learning Engineering Nanodegree (Udacity)

About You / Your Interest in the Journey

Have you used a version control system before?

Which version control systems?

How did version control help in your project?

What would you like to gain from tonight's "Git and GitHub" discussion?

Overview of Version Control and Git

What is Version Control?

Version control systems (VCS) are a category of **software tools** that help a team **manage changes to source code** over time.

Version control software keeps **track** of **every modification** to the code.

If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

Source: https://www.atlassian.com/git/tutorials/what-is-version-control

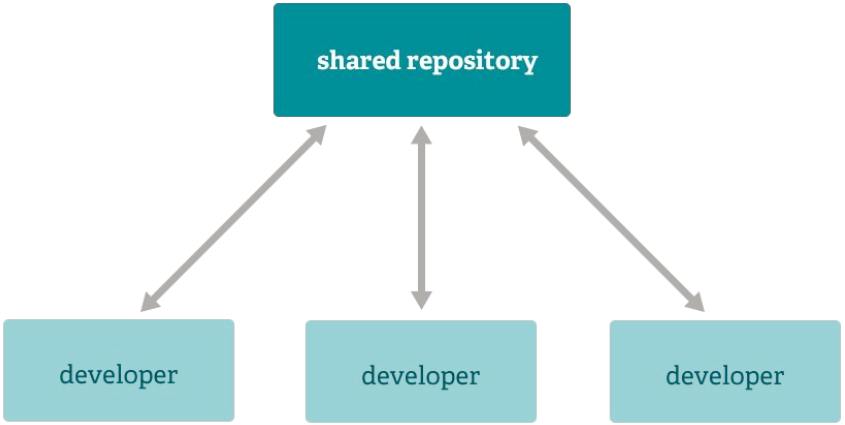
Why Learn Git?

- Code Versioning
  - Access to prior versions
  - Compare and review changes
  - Understand what happened
- Code Collaboration
  - Work
  - Open source
  - Personal projects
- Distributed VCS Benefits
  - Code and commit locally
  - Distributed repo

Why Learn Git?

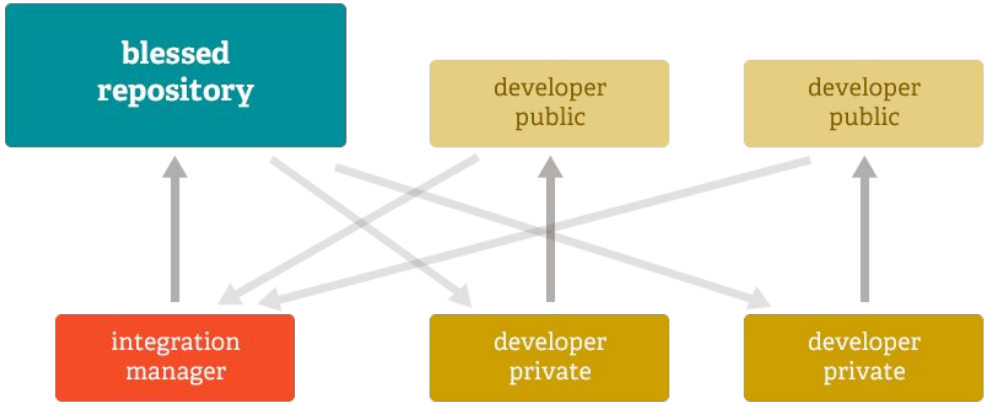
- Frictionless context switching
  - git stash ; git checkout other_branch ; <do_work /> ; git checkout prev_branch ; git stash apply
- Disposable, or not, code experimentation
- Multiple work patterns possible
  - Centralized (like Subversion or CVS)
  - Integration managed
  - Dictator, lieutenants, and developers (large project such as Linux kernel development)
- Free and open source tool
- Interface with SVN repos (or convert to Git ones)

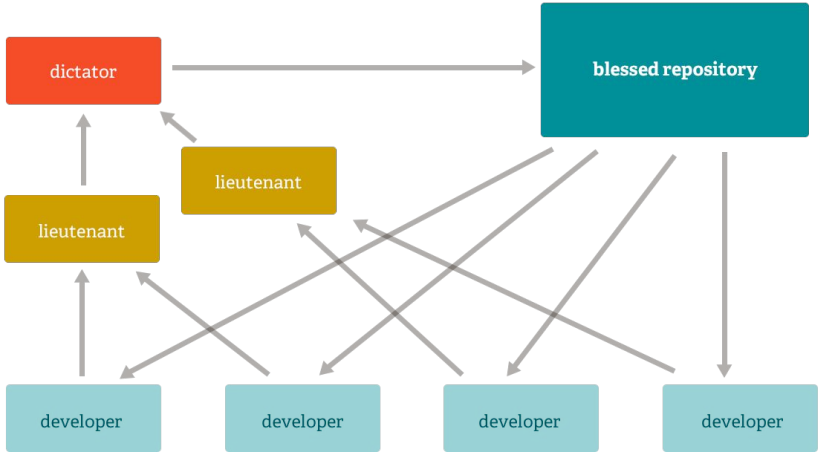
Sources: https://www.git-tower.com/learn/git/ebook/en/desktop-gui/basics/why-use-version-control, https// git-scm/about, https://git-scm.com/book/en/v1/Git-and-Other-Systems-Git-and-Subversion



Central or shared repository model

Integration manager repository model





Dictator, lieutenants, and developers repository model

https://git-scm.com/about/distributed

Metaphor for Git

.. think of it as a code timeline management utility.

Snapshots of

- A point in time
- Point of interest in a project's history

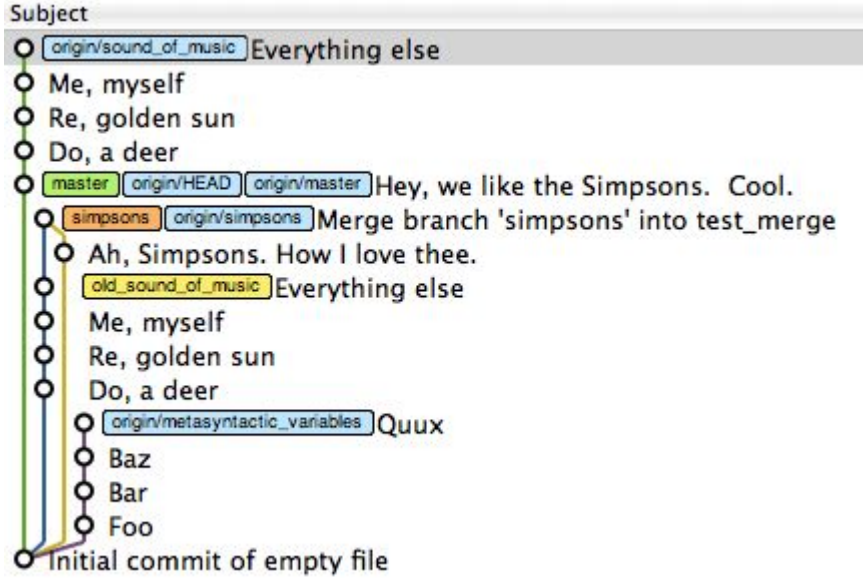
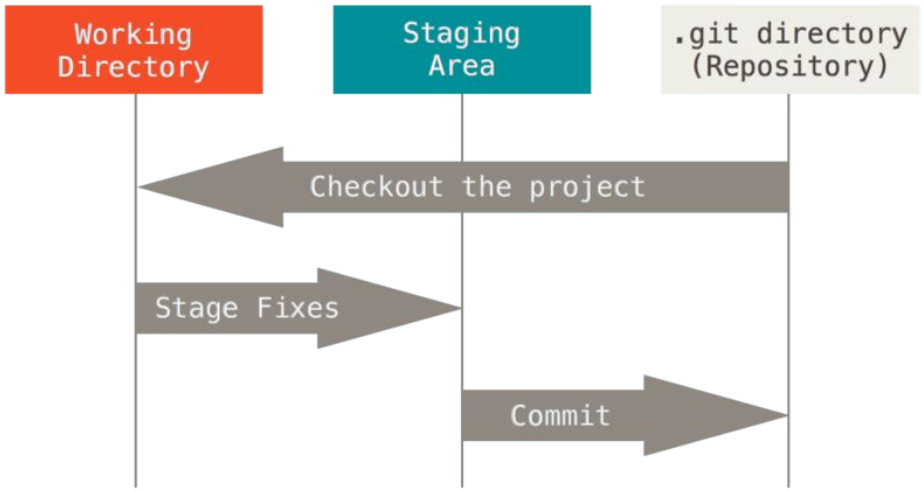


Image: http://think-like-a-git.net/assets/images2/gitx-visualization.png

Git Basics

Three States of a File

- **Committed** - the data is safely stored in your local database.
- **Modified** - have changed the file but have not committed it yet.
- **Staged** - have marked a modified file to go into the next commit snapshot.



Source: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Basic Git Commands

$ **git init**

Creates a new git repository in the present working directory

$ **git status**

Shows the status of files in the repo -- indication if changed or untracked

$ **git add <filename>**

Stages a file for the next commit snapshot

$ **git commit**

Commits the staged files

Git Commands for Remote Repo Interactions

$ **git clone <remote repo URI>**
Repos are cloned (not checked out) locally, or over https, ssh, git protocol

$ **git fetch**
$ **git merge**
Fetch retrieves information about remote repo changes. Merge ‘merges’ in the changes. (git pull does these actions in a single command; use at your own discretion.)

$ **git push**
Pushes local repo committed changes to the remote repo where they are now included.

Git Commands Exercise

Setup Git Environment

Setup name and email configuration

$ **git config --global user.name “Your Name”**

$ **git config --global user.email “your.email@domain.com”**

If collaborating among developers using Windows (CR) and Mac/Unix (CRLF) , may set

$ cd <git-repo>

$ echo “*.py text” >> .gitattributes

to avoid line ending differences

Git Immersion Lab 1 http://gitimmersion.com/lab_01.html

Mind Your End of Line https://adaptivepatchwork.com/2012/03/01/mind-the-end-of-your-line/

Create a Project

$ **mkdir hello**
$ **cd hello**
$ **echo “print(‘Hello, World’)” > hello.py**

$ **git init**
Initialized empty Git repository in .../hello/.git/

$ echo “*.py text” >> .gitattributes # optional command

$ **git add hello.py**
$ **git commit -m “First Commit”**

Git Immersion Lab 3 http://gitimmersion.com/lab_03.html

Checking Repo Status

$ **git status**
On branch master
nothing to commit, working directory clean

Git Immersion Lab 4 http://gitimmersion.com/lab_04.html

Making Changes

$ **echo "print('I\'m excited to use git.')" >> hello.py**

$ **git status**
On branch master
Changes not staged for commit:
  (use “git add <file>...” to update what will be committed)
  (use “git checkout -- <file>...” to discard changes in working directory)

        modified: hello.py

no changes added to commit (use “git add” and/or “git commit -a”

Git Immersion Lab 5 http://gitimmersion.com/lab_05.html

Staging Changes

$ **git add hello.py**
$ **git status**
On branch master
Changes to be committed:
 (use “git reset HEAD <file>...” to unstage)

 modified: hello.py

Separating staging and committing steps in git allow fine-tuning the commit process. Additional files or changes can be staged. Then staged items may be committed.

Git Immersion Lab 6 http://gitimmersion.com/lab_06.html

Committing Changes

If the **-m** flag is omitted from **git commit** command, then the commit process allows one to interactively edit a comment for the commit.

$ **git commit**
|
Please enter the commit message for your changes. Lines starting
with '#' will be ignored, and an empty message aborts the commit
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
modified: hello.py

Git Immersion Lab 8 http://gitimmersion.com/lab_08.html

Changes, Not Files

$ **echo "print('Salt Lake PyLadies is an awesome meetup\!')" >> hello.py**
$ **git add hello.py**

$ **echo “print(‘A topic for a future meetup could be …’)” >> hello.py**
$ **git status**
On branch master
Changes to be committed...
        modified: hello.py
Changes not staged for commit…
        modified: hello.py

Git Immersion Lab 9 http://gitimmersion.com/lab_09.html

Changes, Not Files -- File Diffs

$ **git diff hello.py**

…
+ print(‘A topic for a future meetup could be …’)

Git will prefix line changes with ‘-’ for a deleted line or prior version of code. Lines prefixed with ‘+’ display the current code change.

$ **git config --global color.ui auto**
$ **git diff --color-words hello.py**
If ‘git color.ui’ is set, then ‘git diff --color-words’ shows a word-by-word diff of code

https://chris.frederick.io/2013/09/04/git-tip-of-the-day-color-words.html

Changes, Not Files -- Finish Commits

$ **git commit -m “Added PyLadies meetup”**
$ **git status -s**
M hello.py

$ **git add .**
$ **git status**
On branch master
Changes to be committed...
        modified: hello.py
$ **git commit -m "Added a topic suggestion"**

Git Immersion Lab 9 http://gitimmersion.com/lab_09.html

History

$ **git log**

$ **git log --stat**

$ **git log --pretty=oneline**

See man git-log for details on various options

$ **git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short**

%h - hash of commit

%d - decorations on commit

%ad - author date

%s - comment

%an - author name

--graph - displays commit tree as an ASCII graph

--date=short - short date format

Git Immersion Lab 10 http://gitimmersion.com/lab_10.html

Alias

To set up the previous command via an alias as **hist**

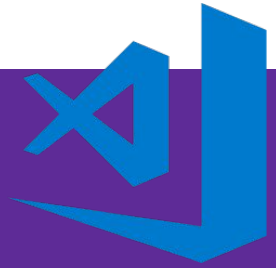
$ **git config --global alias.hist "log --pretty=format\"%h %ad | %s%d [%an]\" --graph --date=short"**

$ **git hist**

```
*   format4ffc0cf 2017-09-18 | Merge pull request #90 from willbeason/willbeason
-patch-1 [Allen Downey]
|\
| * formatd34641c 2017-09-17 | Fix link to issues tab [Will Beason]
|/
*   format9542d7a 2017-08-31 | Merge pull request #75 from imbolc/patch-1 [Allen
 Downey]
|\
| * format817694f 2017-07-22 | Make assertion message more valuable [Imblc]
* |   format23b911e 2017-08-31 | Merge pull request #87 from gbremer/readme_upda
te [Allen Downey]
|\ \
| * | formata949d44 2017-08-30 | #82 Adding seaborn and lifelines of libraries t
o install when setting up an environment [Grant Bremer]
|/ /
* |   format6c3b1dd 2017-08-30 | Merge pull request #86 from gbremer/py3_fixes [
Allen Downey]
```

Git repo history from https://github.com/AllenDowney/ThinkStats2

Tools: TortoiseGit, PyCharm, VSCode



TortoiseGit

Git tool providing Windows Explorer integration.

Simple way to move from storing files in a directory to a git repo for managing changes.

Overlay icons show git status of files.

Another option to understand git: install, then work through previous exercises.

https://tortoisegit.org and https://tortoisegit.org/about/screenshots/

PyCharm

Create a new project

VCS > Import into Version Control > Create Git Repository
File listing will open for project directory; simply click [OK]

VCS > Browse VCS Repository > Show Git Repository Log

Create a new file with some text. Then right click on file tab
Git > Add
Git > Commit

Notice the Version Control Log window has been updated with the recent commit.
https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html

VSCode

Click Git icon to choose to put workspace under Git by using [Initialize Git Repository].

The UI will show indicators for
(# changed files)
U - Untracked files
A - Added files

Document the commit in the Commit Message box, then Ctrl+Enter to actually commit.

If the UI uses a Share icon rather than Git one, “git.path”:<location> may need to be specified in VSCode Settings.

https://www.codeproject.com/Articles/1177391/How-to-Initialize-a-Git-Repository-using-Visual-St

GitHub Tutorial

Why GitHub?

- Start participating in an open source project
  - Adding documentation
  - Submitting a bug report
  - Submitting a pull request (PR) to fix a bug or add a feature
- Practice a programming language on exercism.io
  - http://exercism.io/languages/python
  - Or improve existing exercises or add new ones at https://github.com/exercism/python
- Share or save dotfiles
  - .vimrc https://github.com/thoughtstream/Damian-Conway-s-Vim-Setup
  - .zshrc https://github.com/robbyrussell/oh-my-zsh

“How Can I Contribute” by Lucy Wyman http://slides.lucywyman.me/how-can-i-contribute.html#7 (with her permission)

Hello World - GitHub Guides Exercise

First, sign in to your GitHub account, https://github.com/

Next, open a new browser tab to go to https://guides.github.com/activities/hello-world/

Hello World - Step 1. Create a Repository

1. In the upper right corner, next to your avatar or identicon, click [+] and then select **New repository**.
2. Name your repository hello-world.
3. Write a short description.
4. Select **Initialize this repository with a README.**
5. Click **Create repository**.

Questions?

https://guides.github.com/activities/hello-world/#repository

Hello World - Step 2. Create a Branch

1. Go to your new repository hello-world.
2. Click the drop down at the top of the file list that says **branch: master**.
3. Type a branch name, readme-edits, into the new branch text box.
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.

https://guides.github.com/activities/hello-world/#branch

Hello World - Step 3. Make, commit changes

1. Click the README.md file.
2. Click the pencil icon in the upper right corner of the file view to edit.
3. In the editor, write…
4. Write a commit message that describes your changes.
5. Click **Commit changes** button.

https://guides.github.com/activities/hello-world/#commit

Hello World - Step 4. Open a Pull Request

1. Click the **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button.
2. In the **Example Comparisons** box, select the branch you made, readme-edits, to compare with master (original branch).
3. Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.
4. When you're satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button.

https://guides.github.com/activities/hello-world/#pr

Hello World - Step 5. Merge the Pull Request

1. Click the green **Merge pull request** button to merge the changes into master.
2. Click **Confirm merge**.
3. Go ahead and delete the branch, since it’s changes have been incorporated with the **Delete branch** button in the purple box.

https://guides.github.com/activities/hello-world/#merge

Forking Projects - GitHub Guides

GitHub has simple guides to other common scenarios available at https://guides.github.com/

Next, launch your browser to https://guides.github.com/activities/forking/ for the next exercise.

Pro Tip: find a GitHub repo for an open source project you are interested in and fork it.

Super Pro Tip: look through the project's issues for 'first commit' tagged issues

Or checkout https://www.firsttimersonly.com/

Pull Requests Involve Humans

- Creating a Pull Request
  - Consider breaking the feature change into manageable chunks, easy to review
  - Add comments in PR to explain, for a specific line, the ‘why’ of code choices
- Reviewing a Pull Request
  - Be empathetic
  - Use the PR as an opportunity to teach and to learn

Any feedback or suggestions can be an opportunity to improve: code or communication

Source: https://blog.usejournal.com/the-art-of-humanizing-pull-requests-prs-b520588eb345?gi=8feeb2a060

Backwards Time Traveling

Researching and Resolving Problems

$ **git blame <filename>**

$ **git bisect ...**

$ **git revert <commit-hash>**

$ **git cherry-pick <commit-hash>**

Visual guide to resolving issues
http://justinhileman.info/article/git-pretty/git-pretty.png

Questions?

Other Resources

- Git --distributed-even-if-your-work-flow-isnt https://git-scm.com
- Git Cheat Sheet
  - https://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf
  - https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet
- GitHub Git Reference https://git.github.io/git-reference/
- Visual Git Refererence https://marklodato.github.io/visual-git-guide/index-en.html
- Stack Overflow [git] https://stackoverflow.com/questions/tagged/git
- Pro Git book (free) https://git-scm.com/book/en/v2
- Introduction to Git with Scott Chacon of GitHub (1:22:11) https://www.youtube.com/watch?v=ZDR433b0HJY

What's Next?

- Find an Open Source project relevant to you and contribute to it in some way
- Python Devs Git Bootcamp https://devguide.python.org/gitbootcamp/
- Learn about git internals https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain
- Check out http://gitready.com to expand your 'git vocabulary'
- Watch some video training
  - Git Basics http://git-scm.com/videos
  - Best practices using Git and GitHub https://services.github.com/on-demand/resources/videos/
  - GitHub Training and Guides https://www.youtube.com/channel/UCP7RrmoueENv9TZts3HXXtw
- Explore BitBucket Git Tutorials https://www.atlassian.com/git/tutorials

What's your idea for where to travel and explore next?