SKILLPILLS

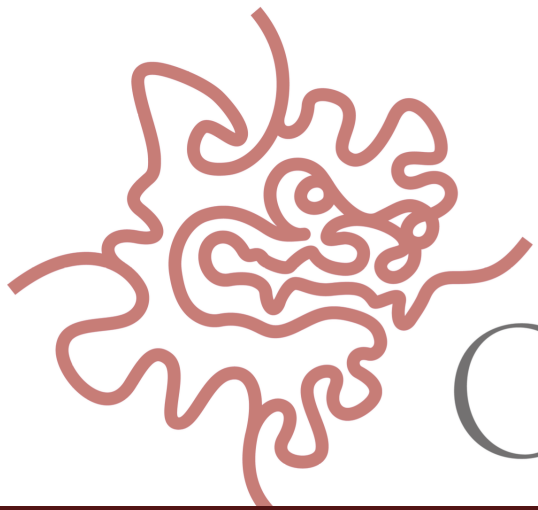
Skill Pill: Introduction to Git and Version Control

Lecture 1: Git ready!

Christopher Buckley

Okinawa Institute of Science and Technology

November 19, 2020



OIST

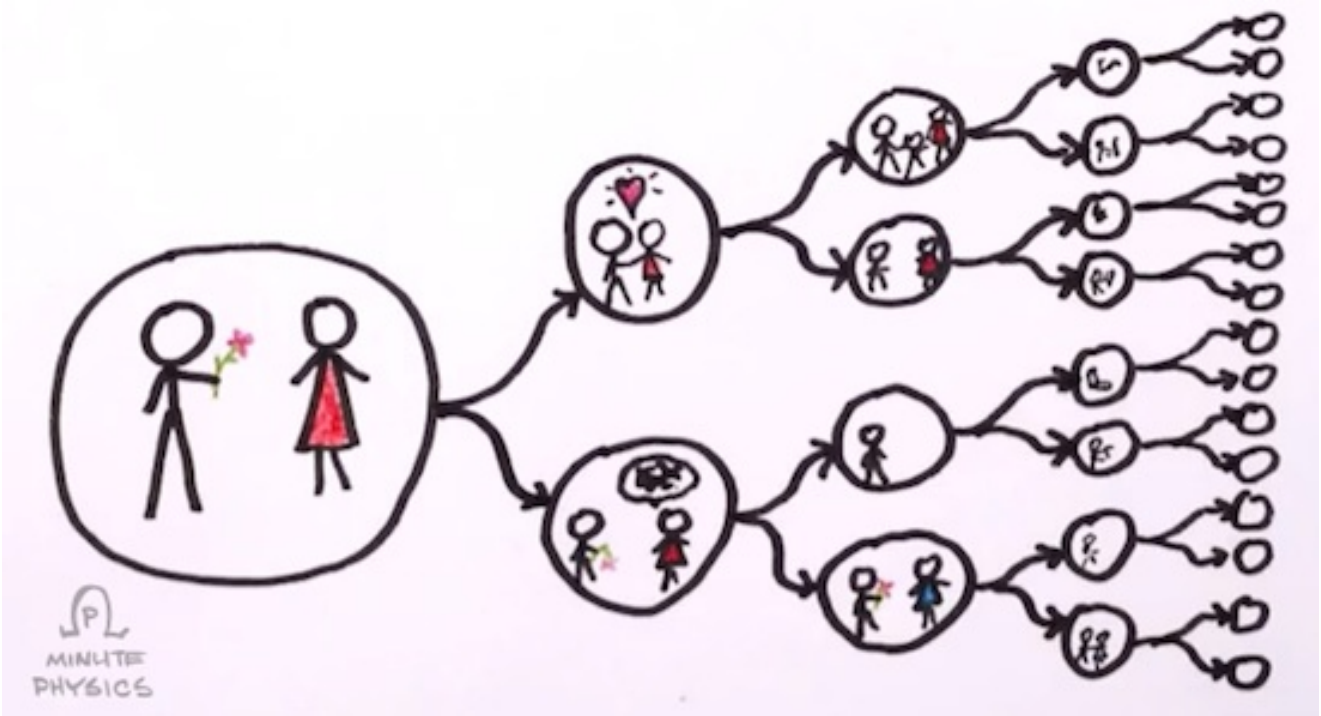
Slides by James Schloss, 2016

Overview

SKILLPILLS

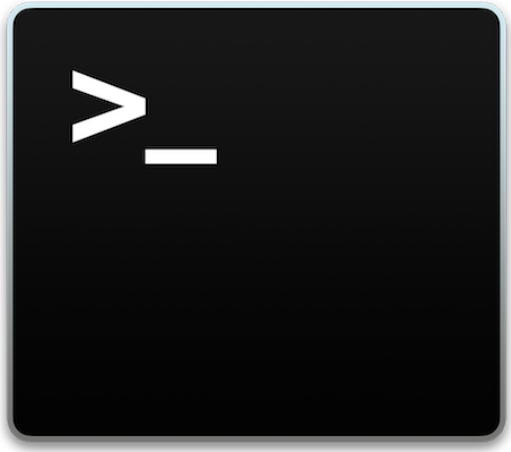
1. What is Version Control
2. Terminal Talk
3. Git basics
   - Local code
   - Nonlocal repos / github
4. Working alone

Version Control



- Version control is a method that allows you to control different versions of things.
- Version control stores history and allows restoration to specific points in that history.

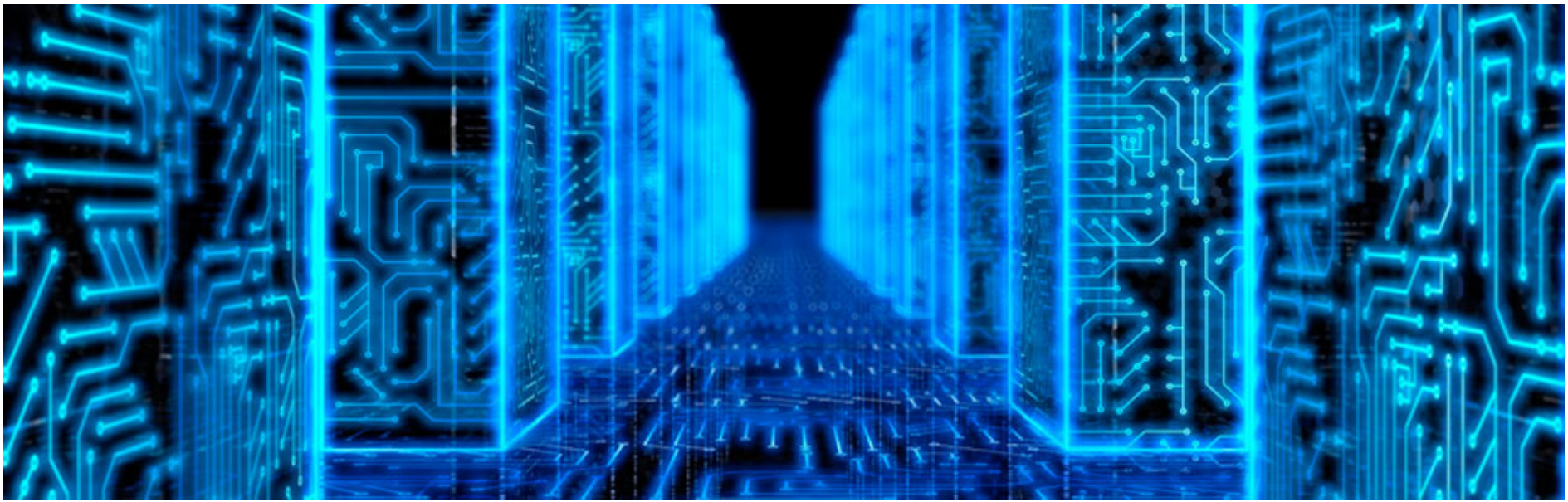
- There are multiple GUIs available for Git, such as one from GitHub called the **GitHub Desktop**. We will not be using this for ~~religious~~ perfectly scientific reasons.
- These reasons primarily revolve around flexibility and improved understanding of the Git tools.
- Everything we do will be usable on Deigo.
- The **Pro Git** book is available online at **git-scm.com/book**
- There is a cheatsheet for Git available here: **https://www.git-tower.com/learn/cheat-sheets/git**



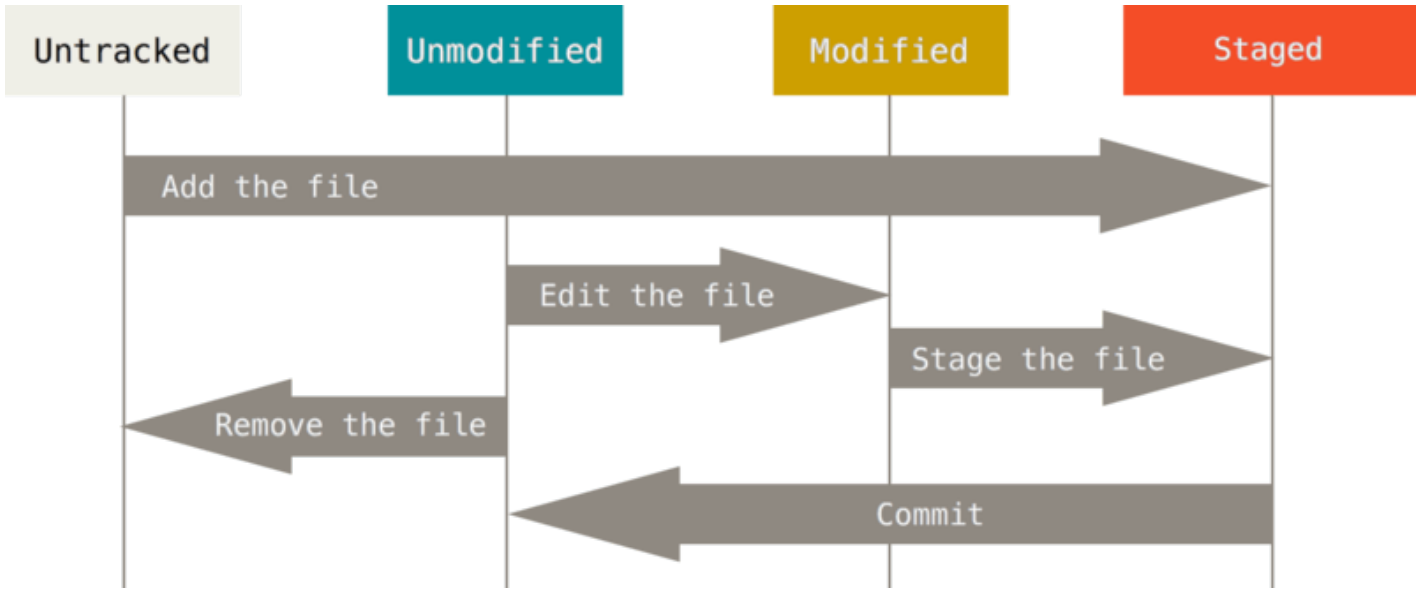
Your first repository

SKILLPILLS

- A **repository** is a place to store code.
  - There are many sites to host your repository on (github, bitbucket), including your own local machine.
  - All of the essential parts of your repository can be found in the **.git** directory
  - GitHub (a website hosting Git repositories) $\neq$ Git (a set of tools for creating and managing those repositories).



File Lifecycle



- A new file is initially **untracked**
- When you use **git add**, it moves to the staging area and becomes **staged**
- After being committed (using **git commit**), a file is up-to-date and considered **unmodified**
- Changing a file makes it modified, but doesn't add it to the staging area

Cleaning the stage

Finally, what is actually happening with your commits under the hood?

- Git has a staging area before commits that can be checked with **git status**. Anything in green is staged.
- If you wish to unstage the commit, simply type **git reset**.
- **git reset** will work for individual files and you may go back to any commit in the history.

```
git reset HEAD~1
```

- If you wish to undo a commit entirely, use the **git revert** command.
- **git clean** (with appropriate flags!) will remove any untracked files.

Quick Exercise

SKILLPILLS

EXERCISE

1. Stage a commit
2. Unstage the commit
3. Make a commit
4. Undo the commit (**DON'T DO THIS AFTER YOU PUSH!!!!!!11111!!!11!!**)

The local repo

Let's **git** started.

- To initialize a git repository, simply type **git init** in a directory (preferably empty for now)
- This creates a folder **.git/**, where all your repository information is held.
- Git tracks **commits**. Check these commits with **git log**.
- **git status** checks any changes since the last commit.
- **git add** adds new files.
- **git commit** commits anything in the *staging area* - git status shows these files in green by default.

Quick Exercise

SKILLPILLS

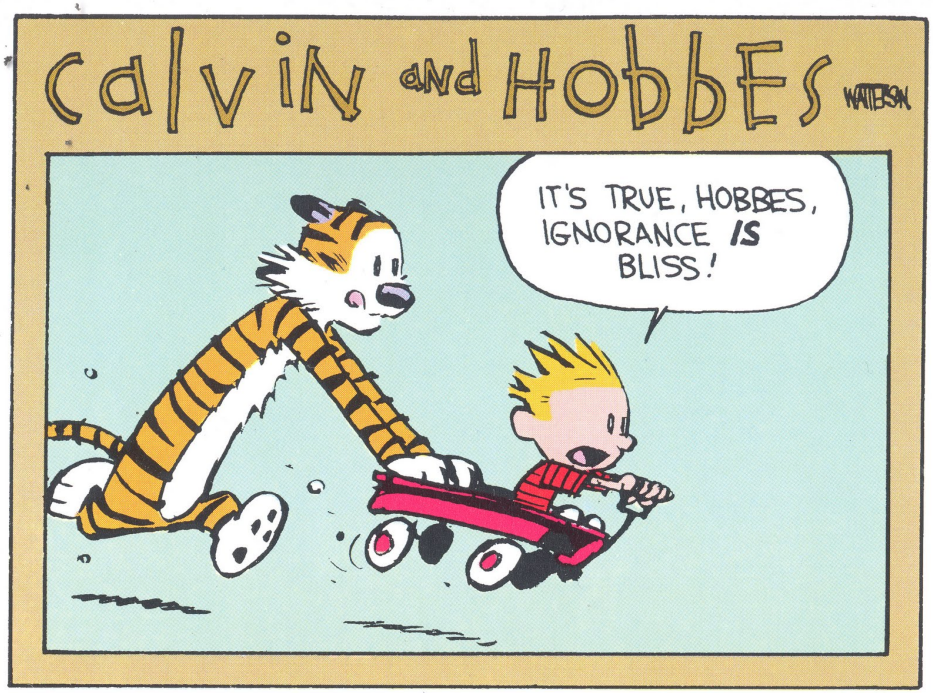
EXERCISE

1. Open a terminal
2. Create a new directory and run **git init**
3. Create a file and run **git status**
4. Use a combination of **git add** and **git commit** to add a new file to the git repository.
5. Check the **git log**.

Ignorance is bliss

- Keep your repository clean! Do your best to commit as few images and data files as possible!
- You can do this by ignoring certain file extensions in a **.gitignore** file.
- Great templates for projects of many types found at **https://github.com/github/gitignore**

```
# Example gitignore
    configuration
*.log
*.tar
*.gz
*.exe
*.dat
*.lvlps
```



Quick Exercise

EXERCISE

1. Touch multiple files with various extensions, one of which should be **.dat**.
2. Ignore the **.dat** file, but commit all the others.
3. Be sure to write a clear message describing what you did.
4. Check the **git log**

Now we move to the fun* stuff: working with **online repositories**.

- For this, we will be using **github**.
- We'll begin by creating a GitHub repository using the website.
  - If we're working on a project that's already hosted on a remote Git server, we can skip this step.
- Next, we use **git clone** to download a copy.
- From here, you can do the following:
  - **git push** to push any changes you may have to the online repository.
  - **git pull** to take any changes from the repository.

*Here, the word *fun* is subject to interpretation.

EXERCISE

1. Create a new GitHub repository using a browser.
2. Clone the new repository* to our local disk:

```
git clone git@github.com:oist/skillpill-git.git
```

or

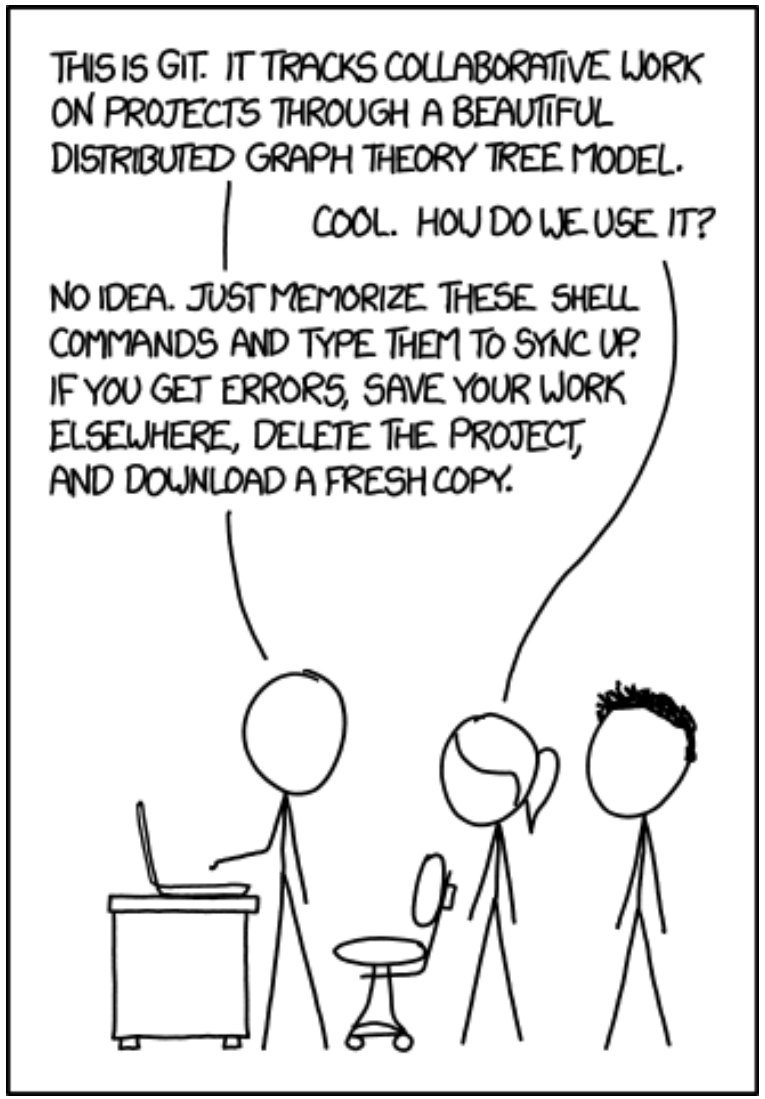
```
git clone https://github.com/oist/skillpill-git.git
```

3. Make some simple commits and test the process of **push**ing and (with the help of a partner) **pull**ing stuff from that repo.

*The examples here show cloning the SkillPill Git repository - replace the links as appropriate!

What it will feel like...

- git is not intuitive to start with, but it's a powerful tool for storing and restoring history, and working collaboratively with other people.
- The more you use it, the more you will like it. Think Stockholm syndrome.
- Operations that you use frequently will become easy.
- Operations you use infrequently, you can Google!



Write clear commit messages!

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Checking out your versions

We now know how to work with both local and online repositories, but what about using different versions?

- **git checkout** allows you to view the repository at any commit (found with **git log**).
- You may also checkout specific files like so:

```
git checkout a1e8fb5 hello.py
```

- Note that the most recent commit is **HEAD** and the one just before that is **HEAD~1**
- This command will be used later, so keep it in mind!

Final Comments

SKILLPILLS

- git is weird. It's not intuitive, but it's the best way to collaborate with people on open projects.
- It's also great even if you don't collaborate!
- Whenever you are using git, think about other people and how they will perceive your comments. **Would you be able to understand your own cryptic commit messages?**
- You will make mistakes. Don't worry about it. Your entire history is backed up already. Learn from your mistakes and don't make them again!
- Read error messages carefully - they can be useful/informative/instructive.