GitHub

Using version control

You probably already use some form of version control if you have files that are named like this:

- `zmays_manuscript-vers1.docx`
- `zmays_manuscript-vers3_CH.docx`

This organization can work well for keeping track of your manuscript, but it doesn't scale well to bioinformatics projects

Git is a *version control system* that helps manage different versions of collaboratively edited code.

Git is tricky to learn but knowing the 5 or so core commands will get you through 90% of situations

For those other sticky situations: stackoverflow is your friend

Not convinced?

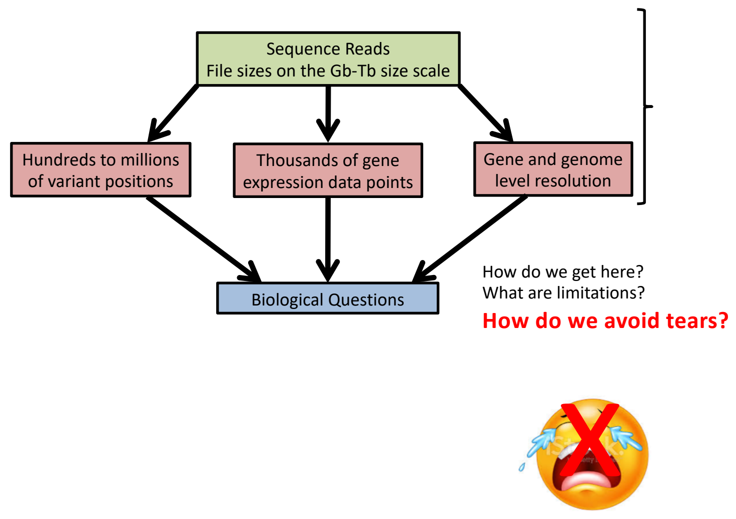
Imagine you keep a lab notebook in pencil, and each time you run a new PCR you erased your past results and jot down the newest ones. This is functionally no different than changing code and not keeping a record of past versions.

Git commits allows you to easily reproduce and rollback to past versions of analysis and is an essential part of proper documentation.

For example, suppose you find that 14% of your SNPs fall in coding regions in one stretch of a chromosome. This is interesting, so you cite it in your paper and make a commit.

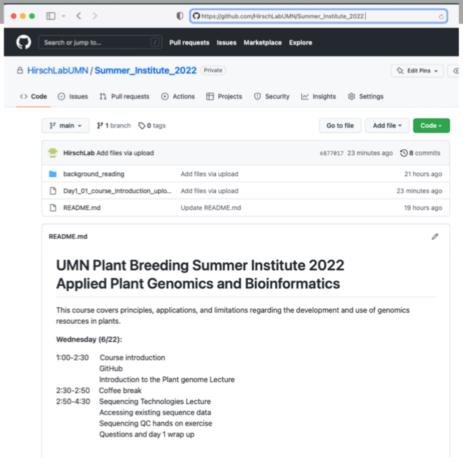
Two months later, you've forgotten the details of this analysis, but need to revisit the 14% statistic. Much to your surprise, when you re-run the analysis code, this changes to 2%.

If you've been using Git you can easily look at the history of code changes and pinpoint the cause of the discrepancy.


Sequence Reads
File sizes on the Gb-Tb size scale
Hundreds to millions of variant positions
Thousands of gene expression data points
Gene and genome level resolution
Biological Questions

How do we get here?
What are limitations?
How do we avoid tears?

How we will use GitHub this week…



HirschLab Add files via upload 6877017 23 minutes ago 8 commits

background_reading	Add files via upload	21 hours ago
Day1_01_course_Introduction_uplo...	Add files via upload	23 minutes ago
README.md	Update README.md	19 hours ago

README.md

UMN Plant Breeding Summer Institute 2022
Applied Plant Genomics and Bioinformatics

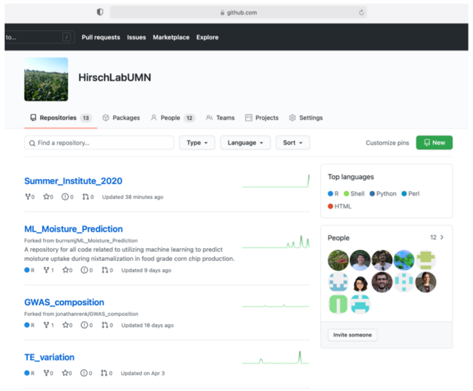
This course covers principles, applications, and limitations regarding the development and use of genomics resources in plants.

Wednesday (6/22):

```
1:00-2:30   Course introduction
            GitHub
            Introduction to the Plant genome Lecture
2:30-2:50   Coffee break
2:50-4:30   Sequencing Technologies Lecture
            Accessing existing sequence data
            Sequencing QC hands on exercise
            Questions and day 1 wrap up
```

All course content is public at https://github.com/HirschLabUMN/Summer_Institute_2022

Using github beyond this course



Summer_Institute_2020
0 0 0 0 Updated 38 minutes ago

ML_Moisture_Prediction
Forked from burnsmj/ML_Moisture_Prediction
A repository for all code related to utilizing machine learning to predict moisture uptake during nixtamalization in food grade corn chip production.
R 1 0 0 0 Updated 9 days ago

GWAS_composition
Forked from jonathanrenk/GWAS_composition
R 1 0 0 0 Updated 18 days ago

TE_variation
R 0 0 0 0 Updated on Apr 3

Create a github account at github.com

Let's see a live example…

README Files

- Placing README files in each of the main directories for your project can save a lot of time and confusion.
- What's in the directory and how the data got there
- Even if you think you could remember all relevant information about your data, it's much easier to write it down- that way your advisor, collaborator, colleague can all use your data.

What to document

What to Document
- **Methods and workflows including full command lines**
- Document the origin of all data
- Document when you downloaded data
- Record data version information (i.e. Ensembl Release 33)
- Describe how you downloaded the data (wget, SQL query, etc.)
- Document the versions of the software you ran

Plain text is preferred because:
- It can easily be read, searched, and edited directly from the command line
- It lacks complex formatting which can create issues when copying and pasting commands from your documentation back into the command line
- It is more future-proof than other formats
- It can be tracked using version control

Getting started with git

First tell git who you are:

```
$ git config --global user.name "Sewall Wright"
$ git config --global user.email "swright@adap-
tivelandscape.org"

$ git config --global color.ui true
```

Git Repository

A *repository* is a directory that’s under version control

Contains both current working files and snapshots of the project at certain points in time - snapshots are known as *commits*

Working with Git is all about creating and manipulating these commits: creating commits, looking at past commits, sharing commits, and comparing different commits.

Initializing a Repository

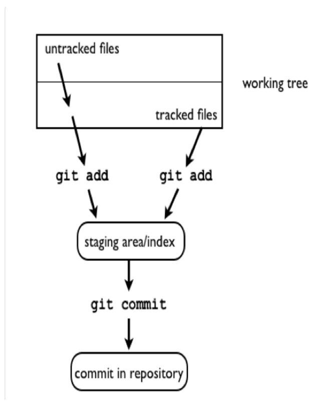
There are two ways of starting a repository:

-Use **`git init`** on an existing directory
This will created a hidden directory .git. Don’t mess with the files in this directory, these are needed so Git can function properly behind the scenes

-The other way is to **`git clone`** an existing repository
You can clone an existing repository from anywhere but most commonly you’ll do so from somewhere like GitHub or Bitbucket.

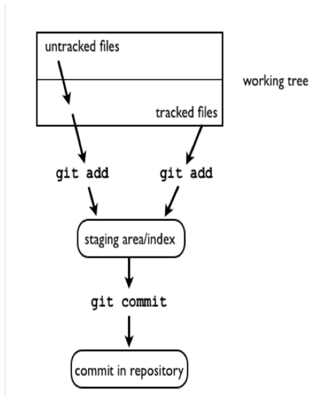
This can also be done online

Adding files to a repository



- Even though you've initialized a directory as a repository, Git doesn't automatically begin tracking every file in the directory
- You need to tell Git which files to *track* using the subcommand **`git add`**
- Before tracking a file, use the command **`git status`** to check Git's status of the files in our repository.

Committing files to a repository



- *tracked* vs. *staged* files
- A file that is tracked means Git knows about it
- A staged file is not only tracked, but its latest changes are staged to be included in the next commit
- Once a file is staged, it is ready to be *committed*

```
$ git commit -m "initial import"
 2 files changed, 1 insertion(+)
 create mode 100644 README
 create mode 100644 data/README
```

- A shortcut that will stage and commit together:
  - **`git commit`** `–a –m  "initial import"`
  - **`git push`**
- Git wants to be in charge when it tracks files, using `mv` or `rm` will confuse Git, therefore we use `git mv`

Getting files from the past

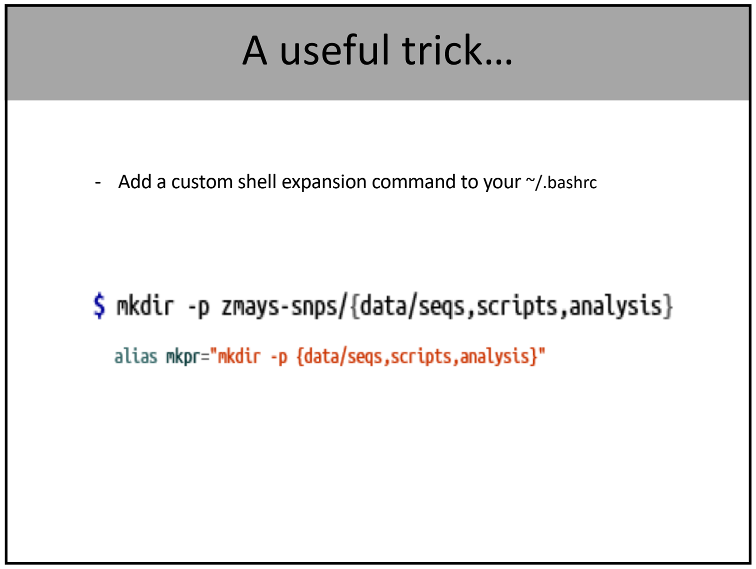
- Suppose you accidentally overwrite your current version of README.md by using > instead of >>.

```
$ echo "Added an accidental line" > README.md
$ cat README.md
Added an accidental line
$ 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:   README.md
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Directory Structure

- Creating a well-organized directory structure is the foundation of a reproducible bioinformatics project
- Develop a project organization scheme that works for you and stay consistent

```
$ mkdir zmays-snps
$ cd zmays-snps
$ mkdir data
$ mkdir data/seqs scripts analysis
$ ls -l
total 0
drwxr-xr-x  2 vinceb  staff   68 Apr 15 01:10 analysis
drwxr-xr-x  3 vinceb  staff  102 Apr 15 01:10 data
drwxr-xr-x  2 vinceb  staff   68 Apr 15 01:10 scripts
```



A useful trick…

- Add a custom shell expansion command to your ~/.bashrc

```
$ mkdir -p zmays-snps/{data/seqs,scripts,analysis}
```

```
alias mkpr="mkdir -p {data/seqs,scripts,analysis}"
```