The Command Line Interface (CLI)

- CLI for Windows is Git Bash; for Linux/OS X it's terminal
- **Path** to working directory is path from root directory to working directory
    - pwd returns path
- CLI commands are in format *command flags arguments*
    - Flags are options given to trigger certain behaviors; usually preceded by –
    - Arguments are what commands will modify or use
- *ls* lists files in directory;
    - *-a* flag lists both hidden and unhidden files and directories
    - *-l* flag lists details
    - Can combine these flags, i.e., *ls -al*
- *touch filename* creates empty file
- *cp file_to_copy path_to_copy_to* copies a file
    - To copy directories, use *–r* flag
- *mv* moves files, deleting original; can use to rename files
- *date* prints today's date

Git and GitHub

What is version control?

"Version control is a system that records changes to a file or set files so that you can recall specific versions later." It's a means of managing intermediate files, and is vital when collaborating.

What is Git?

A version control system created by the same people who developed Linux. It stores everything on your computer in local repos

Git Bash is CLI for interacting with Git on Windows

Each commit is tagged with the username of the person who made the commit, so after installation, must assign a username with:

1. *git config --global user.name "Name"*
2. *git config --global user.email "email"*
3. Confirm these changes: *git config --list*
4. *Exit*

Don't need GitHub to use Git; git=local, GitHub=remote

GitHub allows you to:

1. Share repositories (repos) with others
2. Access other users' repos
3. Store remote copies of repos in case something happens to local ones

To get help, check Git docs (http://git-scm.com/doc) and GitHub help (https://help.github.com), but usually Google and Stackoverflow are faster.

Creating a GitHub Repo

Start a repo from scratch: Go to profile page and click "create a new repo" in upper-right corner, or go to https://github.com/new. Then create a local copy: open Git Bash, create new directory, and cd there; *git init; git remote add origin https://github.com/mg41/reponame.git*

Fork (copy) another user's repo: Navigate to repo page and hit "fork". Then make a local copy ("**clone**"), with *git clone https://github.com/mg41/reponame.git*

NB: this will clone repo into the working directory

NB: you clone after forking; you remote add origin after creating a new repo

Basic Git Commands

The structure of Git:

- **Workspace** is where actually doing work
- **Index** tells git which files to control
- **Local repo** is files that are stored and version controlled on local computer
- **Remote repo**

The basic process is simple. Create new file and **add** to index so Git knows to monitor it. **Commit** changes to file to local repo. **Push** local repo to remote repo.

Adding

- Suppose add new files to a local repo under version control
- Need to let git know that it needs to track these files
    - Add all new files: *git add .*
    - Update tracking for files that changed names or were deleted: *git add –u*
    - Do both: *git add –A*
- Add before committing, because must put the files into the index

Committing

- Suppose you have changes you want to commit, so they're saved as an intermediate version. Use *git commit –m "useful description of what did"*
- This only updates the local repo

Pushing

- Suppose saved local commits that want to update on remote repo; use *git push*

Branches

- Sometimes working on a project with version being used by many people; may not want to edit that version, so use **branches**
- Create a branch: *git checkout –b branchname*
- Check current branch: *git branch*
- Switch to (master) branch: *git checkout master* (NB: *–b* flag used to create branch)

Pull Requests

- If fork someone's repo, or have multiple branches, will both be working separately.
- Sometimes want to merge your changes into the other branch/repo. To do so, need to send pull request by going to the desired branch on GitHub and hitting "compare a pull request" (green button)
- Unique feature of GitHub; not part of Git

Basic Markdown

Textfile w/ .md extension

Headings:

##secondary heading

###tertiary heading

Unordered lists

*first item

*second item

*third item

More at http://daringfireball.net/projects/markdown

R Packages

- Primary place to get packages is CRAN (Comprehensive R Archive Network), with over 5200 packages
- Packages for biological applications can be found at the Bioconductor Project
- To list available packages, use *available.packages()*
    - This will provide a lot of packages; use this code to display a manageable amount
        1. a<-available.packages()
        2. head(rownames(a), 3) ##show names of first few packages
- To install a single package, pass name to *install.packages,* e.g., *install.packages("slidify")*
- To install multiple packages, use character vector: *install.packages(c("slidify", "ggplot2", "devtools"))*
- R automatically handles installation of dependencies
- In RStudio, install by going to Tools menu, then Install Packages submenu

Installing from BioConductor

To get basic installer and basic set of packages run

source(http://bioconductor.org/biocLite.R)

biocLite()

To install other packages, run *biocLite(c("GenomicFeatures", "AnnotationDbi"))*

Loading Packages

- After installing a package, must **load** it to make it available; do this with the *library()* function, e.g., *library(ggplot2)* [NB: package name not in quotes]
- Any dependencies will be loaded first, so must have all dependencies installed.
- To see a list of installed packages, call *search().* After loading a package, functions exported by that package will be attached to the top of the *search()* list.

Installing Rtools

- Collection of tools needed for building R packages in Windows
- Available at http://cran.r-project.org/bin/windows/Rtools
- Download version corresponding to R version—to check the R version, just fire up R and the version will be displayed in the console—then run the .exe
    - Make sure check the box to edit path. Unless have Cygwin installed, leave all other options default.

- Once Rtools installation complete, open RStudio and install devtools. To check if it's already installed, run *find.package("devtools").* If not, install and load it. Verify that it was installed by running *find_rtools(),* which should print TRUE.