# Session 1:

Summary: what is git, its basic settings

Goal: configured and authenticated

# Basics:

1. Version Control System (VCS):

Help us getting rid of “final\_another\_rev \_I’ll\_chop\_off\_my\_hand.doc”

1. Distributed Version Control System (DVCS):
2. **Local** or **centralized** VCS
3. Benefits:

* Flexible, support remote collaboration, different uncentralized workflow
* Robust against server malfunction (retrieve cloned user repository)
* Less dependent on Network/VPN (access cloned repository offline)
* Other systems: Mercurial, Bazaar or Darcs. Not sure whey Git is so popular. Maybe because it is free?

1. GitHub: user-friendly

# Set-up:

Quick pool: how has already set up git

Not sure: use git –version to check.

Note: Books/tutorials tend to talk about Configuration and Authentication later. But it is usually the first problem Git users run into when they try to work remotely.

# Configuration

1. Definition

* Customization. Make Git more specific to certain purposes (system, user, repository).
* Focus on user level – sufficient for the PhD/Master program

1. Check configuration setting

* Two different files: system and user level
* Get where information is stored, open them (start for windows)

1. Change configuration setting:

* Git config + level + option + information
* Make sure the username and email match your GitHub account!
* Change a lot more

# Authentication:

1. Server protocols:

* Local: not good with multi-site work
* SSH and HTTP: use to be equivalent, until HTTP started asking for personal token
* Git: Not as secure as the previous ones

1. Authentication settings

* SSH keys: stored in ~/.ssh

1. Generate New Key:

* Following settings are algorithms
* Prompt open location
* Remember the passphrase

1. Additional notes

* If do not want to use key chain:

1. Omit the corresponding line when adding to ssh
2. Run ssh-add without -apple-use-keychain

* If Git asks for passphrase every time you pull or push: Open SSH agent: eval $(ssh-agent), Add personal SSH key: ssh-add
* General debug for windows:

1. Are you running through bash terminal?
2. Use “start” in place of “open”
3. npm install touch-cli -g

# Session 2: Local command lines

Review: configuration & authentication

Today: focus on working locally in a specific directory/repository.

1. Local workflow

* Working tree: office desktop/working station (some times “working tree clean”)
* Staging area: set aside what is done
* Repository: put it in a folder and store on a shelf/cabinet

Operations: move files from places to places

1. Initialization
2. Start remotely: Clone

Which link you use depends on authentication protocal

1. Start locally

* Create/have a local project, then covert it to a git repository.
* First, make sure you are on the right working directory.
* Git init: initialize the local project as a git repository.
* Add and commit: more later. For now: its saying put everything that’s in the folder on Git
* Experiment (R project, and a script) and use terminal in R studio from now on
* Git commands can be used on it from now on

1. Status of files

* Untracked:
* Tracked: modified/unmodified/staged(done editing, set aside)
* Side note: certain files that is not on git. Git just couldn’t see.

For now just know it’s possible to have files on hard drive but not on git.

* Command lines: used to move files across different status

Note: “commit” is not reverse, but rather reset (start a new round of editing). More explanation later.

1. Git status and Git add:

* Git status: full or simplified
* Git add: add to the staging area/stage files
* Example: 1) Create a new file to illustrate “untracked”, 2) Modify a file to illustrate “modified”

1. Gitignore files:

* Very useful. Git is not efficient for track large files when working remotely
* Create repository on GitHub: default ignore file
* Remember from last time: “touch” creates files. ~/ omitted because we are in the current working directory
* Open and edit gitignore: general expression (ignore \*.RData)
* Ignored files: Git just cannot see them (different with untracked), act like it doesn’t exist.

(git status, RData is not gonna show up under “untracked”)

* Experiment

1. Remove files: git rm

* Move back along the workflow
* Try with df.RData: not gonna work, because it is ignored by Git
* Git rm: remove from your device. Can’t get it back. Use with caution!
* Git rm: cached: move it back to “untracked”. Git still sees it.

If use “add .” it will be add back to git

* Remove from Git: ignore

1. Change commit history

* Create the “NewCode” file back
* Git restore: modify file (use with caution)
* Git restore –staged: does not change file, but change file status
* Example: edit file, then add, then restore stage, the restore

# Collaboration:

Pull = fetch+merge

# View commit history:

* git log, with a lot of options
* But mostly can be done on GitHub, so omitted

# Undo changes:

* Add to last commit
* Unstage changes: restore
* Remove from working tree: reverse changes on local hard drive

Git checkout: Any local changes you made to that file are gone

# Branches

1. Create branches
2. Merge branches and conflicts

* Happens 99% of the time
* Need to solve by hand

# Session 3: Remote command lines

-explain why sometimes people can push/pull with wrong configuration