# Command Line & Git*

Chapter 1

Anna Ziff

R Workflow for Economists

## Contents



*Please contact anna.ziff@duke.edu if there are errors.

# Command Line

The command line is a direct way to type instructions for your computer to immediately execute. Comfort with some basics can help you more easily use R, Git, and any other language or software.

- If you are a Windows user, open GitBash. There are other options, but practicing in GitBash (or your Git application of choice) will help you learn Git later on.
- If you are a Mac user, open Terminal.
- If you are Linux user, open Terminal or your system's shell application.

## Navigating Directories

Print the working directory with `pwd`.

```
pwd
```

List files in the current directory with `ls`. Sometimes there are hidden files (e.g., `.gitignore`). Add the `-a` option to see those as well.

```
ls
ls -a
```

Change directory with `cd`. To change to the root directory, specify `/`.

```
cd /
```

To change to the home directory, specify  .

```
cd
cd ~
```

You can change the directory to a specific path.

```
cd path/to/file
```

Use `..` to indicate one level back.

```
cd ..
cd ../../relative/path/to/file
```

Note that nothing is printed after `cd` unless you encounter an error. You can always check where you are in your file system with `pwd`. If you are using a Windows machine, you may need to use the backslash instead for the file paths.

You can create a folder in the current directory using `mkdir`. The flag `-p` will create `dir/dir1` if those directories do not already exist.

```
mkdir dir
mkdir dir1 dir2 dir3
mkdir -p dir/dir1/dir1a
```

You can move a file to another directory using `mv`.

```
mv file.txt path/to/folder/text.txt
```

The command `mv` can also be used to change the name of a file.

```
mv file.txt newname.txt
```

You can copy a file using `cp`.

```
cp file.txt file_copy.txt
```

With caution, you can delete files using `rm`.

```
rm file.txt
```

Be *very careful* with this command. It is easy to accidentally erase all the files in your computer with the command (DO NOT TYPE THIS!!!!) ~~`rm -r *`~~ (DO NOT TYPE THIS!!!!). It is good practice to check which directory you are in (`pwd`) and slowly remove files and folders one at a time rather than deleting them in batches.

See Canonical Ltd. (2021) for more commands and details. The ones listed above are more than sufficient for this class.

### Vim

Vim is a text editor accessible through the command line. It is often the default text editor for Window's GitBash, Apple's Terminal, and Linux's shell. There are other options (Nano, Emacs, Notepad++), and you are free to choose the one that works best for you. I will highlight the basics of Vim as that is the more universal option.

Open a read-only version of a file in Vim. If the file does not exist, Vim will create an empty file.

```
vim file.txt
```

To edit the file, type `i` and begin editing. To exit without saving, `esc + :q`. To exit with saving (write and quit), `esc + :wq`.

### Naming Conventions

Only use letters (`a-z`, `A-Z`), numbers (`0-9`), underscores (`_`), and hyphens (`-`) in your file and folder names. This will make navigating your file system from the command line much smoother. It is also good practice to have a consistent naming system.

| Name | Example |
|---|---|
| Dash case | `my-file.txt` |
| CamelCase | `myFile.txt`, `MyFile.txt` |
| Snake case | `my_file.txt` |
| Flat case | `myfile.txt` |
| UPPERCASE | `MYFILE.txt` |

### Practice Exercises 1.1

1. Open the command line on your computer. Print your working directory and list the files there.
2. Navigate to where you want to have a folder with the work for this class (e.g., `Documents`).
3. Create a folder called `temp`. Navigate inside the `temp` folder.
4. With one line, create two folders inside the temp folder called `temp1` and `temp2`.
5. Create a file called `test.txt` and save it to `temp/temp1`. Challenge yourself by creating this file from the command line using Vim or another editor. Type `"Hello world!"` and save the file.
6. Move `test.txt` from `temp1` to `temp2`.
7. Make a copy of `test.txt` called `test-copy.txt`. Save it in `temp1`.
8. Delete the file `test.txt`.
9. Confirm you are in the path from exercise 2. Now create the folder in which you will save all your class material. Call it whatever you would like.

## Git

Git is a distributed version control system. It documents the complete history of a project, including an archive of changes and previous versions of files. Economists commonly use Git across sectors.

Git has several advantages over other formal and casual methods of version control. It is a free tool that centrally stores project files with a record of all changes, who made the changes, and why those changes were made. Each change of a file is documented, and once entered in the database, cannot be changed. The disadvantage is that it takes some time and practice to learn how to use Git. This handout describes Git and introduces the essential Git commands.

## Setting Up Git

### Installation

Git must be installed on your local machine, i.e., your personal computer. This section explains the most flexible way to use Git, which is through the command line (a direct place for you to type instructions to your computer). There are many GUIs (graphical user interfaces) available as well. Once you understand how to use Git on the command line, you should be able to learn any GUIs or other Git-related tools easily.

Mac computers should already have Git installed. You will access Git through the Terminal application (this application is how Mac users access the command line). Open Terminal and make sure you have Git installed by typing the below. It is fine if your version is different than mine.

```
git --version
```

```
## git version 2.32.1 (Apple Git-133)
```

If it is not installed, you will get instructions on how to install it.

Machines with Windows operating systems require more setup. Git Bash allows you to use Git in the command line the same way a Mac or Linux user would. The below steps are copied from Matoso (2019).

1. Download the Git installer.
2. Execute the file with the default options in the "Select Components" page.
3. The option "Windows Explorer integration > Context menu entries" makes it possible to right click on a folder and open Git Bash to that folder's location.
4. In "Adjusting your path environment" select "Use Git Bash only."
5. Select "Checkout as-is, commit Unix-style line endings." This helps prevent compatibility issues due to different line endings with files from other operating systems.
6. Click "Next" and then "Finish."

You can also use another program like PowerShell or Command Prompt. Git Bash seems to be the preferred application.

See page 17 of Chacon and Straub (2020) for installation instructions for RPM-based and Debian-based distributions (Linux).

### Configuration

Once Git is installed, there are several options to configure options available through the `config` command. You will need to set your user name and email.

```
git config --global user.name "Your Name"
git config --global user.email youremail
```

The `global` option means that this configuration is the same regardless of the project. You can always check your values.

```
git config user.name
git config user.email
```

```
## aziff
## anna.ziff@duke.edu
```

GitHub recently changed its security so that you require a Personal Access Token (PAT) to use Git from the command line.

1. Navigate to the settings of your GitHub account.
2. On the left-hand side menu, scroll all the way to the bottom and click on "Developer Settings."
3. Click on "Personal access tokens."
4. Click on "Generate new token." You may have to sign in with your GitHub credentials.
5. Give the token a name, set the expiration for the appropriate amount of time, and select all scopes. Many of these are applicable to repositories that publish packages or software, but it is easier just to select all of them.
6. Click on "Generate token."
7. Your token will appear on the screen. Copy it into a location where you can easily access it. This can be some note-taking software or a text file on your computer. Once you exit this screen, you will not be able to see this token again.

## Characteristics of Git

- Instead of storing changes to files across versions, Git stores snapshots of the project over time. Figure 1 from Chacon and Straub (2020) provides a visualization of what saving changes would look like. Figure 2, also from Chacon and Straub (2020), shows how Git stores data. Effectively, a picture of the whole project is taken each time you "save" your project.

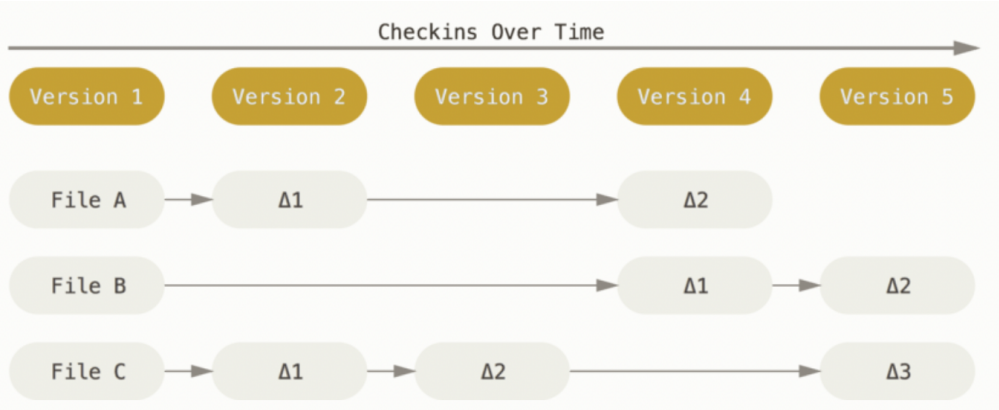


Figure 1: Storing Changes

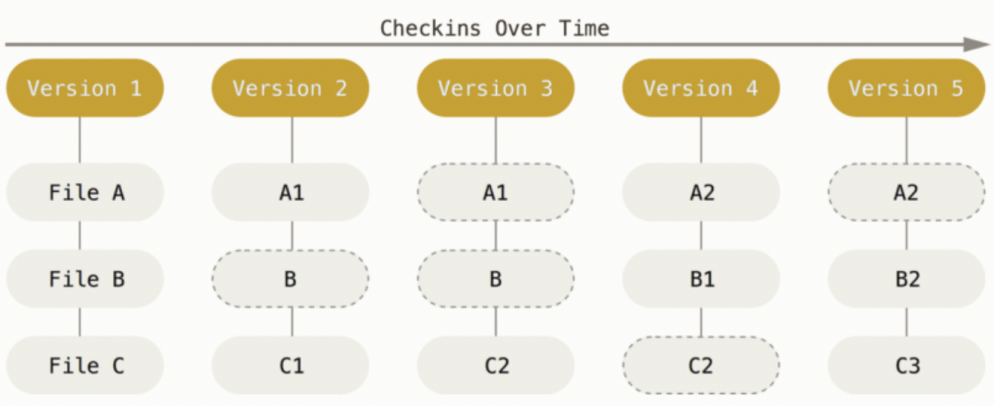


Figure 2: Storing Snapshots

- All the project files and the project's history are stored locally on your computer. This minimizes

reliance on connecting to an external server, aiding speed and stability. It also allows one to keep track of changes even without an internet connection.

- Once "saved," a project's history cannot be changed. This gives Git integrity; You cannot accidentally corrupt or lose information without a warning or error message. Everything in a Git database is checksummed and given an SHA-1 hash, basically a long string of numbers and letters. Here is an example SHA-1 hash: `6c658d1e96acb313eed5e9d13d723275b6479d04`.

## Repositories

A repository is a folder whose contents are under Git's version control. You can turn any folder on your computer into a git repository. See page 245 of Chacon and Straub (2020) for instructions on this. In this class, we will talk about how to set up a Git repository on an external server (in our case, GitHub) and clone it on you local machine. Using a website like GitHub as a server for your Git-controlled repository has its advantages. The ability to control who can access your repository helps streamline collaborations. There are many interactive features on the website to help keep track of issues and changes for the project. Even without collaborators, using GitHub provides a natural backup for your own work.

Once the repository is setup on GitHub, you need to clone it to your local machine. Navigate to the directory where you want to store the repository and clone it there.[1]

```
git clone https://github.com/aziff/R-Mini-Summer2022.git
```

This essentially copies the entire repository, including all tracked changes, to your local machine. There is a hidden directory, `.git`, that stores all the version control information. The presence of this directory is what makes a folder a repository tracked by Git. To clone a repostiory, you will usually need to enter your PAT.

## Save and Track Changes

Each file in the repository can be tracked or untracked. If a file is tracked, then it is backed up in the server's repository and changes can be recorded. If a file is untracked, then Git does not "know" about it and any changes made to it are not stored.

- Tracked Files. A tracked file can be in three states: unmodified, modified, or staged. If a file is unmodified, then the copy of the file on your local machine is identical to the last snapshot of the file in the repository. Once you make changes to the file, it is modified. The file becomes staged once the modifications are recorded to the repository. Figure 3 is from Chacon and Straub (2020) and visualizes these three states.
- Untracked Files. There are certain files that should remain untracked. Because Git takes a snapshot of the whole repository, keeping track of very large or complex files is burdensome. Generally speaking, do not track datasets, PDFs, images, auxiliary files, or Microsoft Office files (more details on this below).

## Basic Git Workflow

Checking the status of your local repository is an important tool to help you navigate the Git workflow.

```
git status
```

If you have untracked files, they will be listed as such after typing `git status`. To track both new files and update the tracking on modified files, you first need to add them.

```
git add file.txt
```

If you run `git status` again, you will now see that there is a new file with "changes to be committed.'' This means that the file is staged. You can also add all the contents of a folder.

[1] This method of cloning is through an HTTP protocol. If you are cloning a repository from GitHub to your local machine, this will be fine. If you want to clone a repository to a server or network share, you may need to use an SSH protocol instead. Follow the directions here for that.

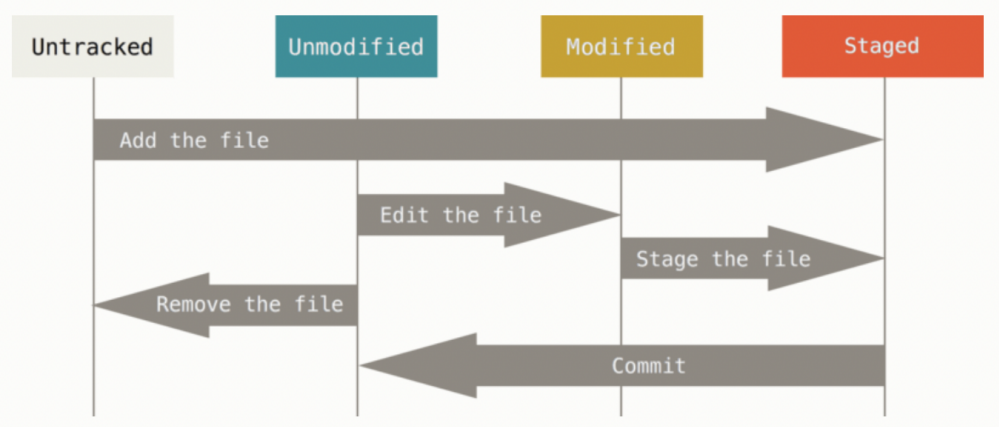


Figure 3: Storing Snapshots

```
git add dir1
```

Here are some shortcuts. The period (`.`) stages every downstream change. The asterisk (`*`) is a placeholder.

```
git add .
git add *.txt
```

Again, you can check the status of the repository to ensure you staged what you wanted. You can use the `-short` (or `-s`) flag if you want to view a more condensed output. To view more details on what specifically was changed in the unstaged file.

```
git status
git status -short
git status -s
git diff
```

You can modify a file further even once it is staged. You will just need to add the file again to keep track of the additional changes.

Once you have staged everything you want, the next step is to commit your changes. Any file that is staged will be included in the commit.

```
git commit
```

This will open a text editor with information about the commit, including what files were newly staged or modified. You can add a message with additional information before quitting the text editor. Exiting the text editor induces the commit to be created with the commit message. It is often convenient to write the commit message inline.

```
git commit -m "Initial commit."
```

Already, this commit provides a record of the current version of the repository. This is useful even without saving on GitHub. However, GitHub is useful for collaboration and to back-up your local machine. To "save" to GitHub, you will need to `push` your commits.

```
git push

# Specifying the remote (origin) and the branch (master)
git push origin master
```

This push will be rejected if anyone else on the project has pushed work that you have not yet integrated

into your files. It is thus good practice to update your files before changing anything.

```
git pull
```

This is good practice even if you are working independently across multiple machines (e.g., you work on your laptop and in the Econ cluster). To review, the workflow should be as follows.

1. Update your local repository: `git pull`
2. Make your changes
3. Stage your changes: `git add`
4. Commit your changes: `git commit -m "Commit message."`
5. Push your changes: `git push`

The command `git log` allows you to view the commit history of your repository. See Chacon and Straub (2020) for details on how to format the output of this. Another option is to view the repository on GitHub's website.

```
git log
```

## Deleting and Renaming Tracked Files

To delete tracked files, use `git rm`. This removes the file from the repository and from the staging area. Once you commit this change, the file will no longer be tracked.

```
git rm results.txt
```

If the file has been modified or it is already staged, you need to add the force flag, `-f`.

```
git rm -f results.txt
```

To remove files from the staging area without deleting the files entirely, use the cached flag. This is an issue that often arises, especially if you forgot to update your `.gitignore` file (more on this below).

```
git rm --cached results.txt
```

If you change the location of a file or change its name, git will view this as deleting the original file and creating a new file with the new name. You can use `git mv` to specify this directly. It is not strictly necessary, but is a convenience function that may be helpful when you want a specific commit message to go with changing a file's location or name.

```
git mv original-name.txt new-name.txt
```

## Should all files be tracked?

In short: no. Keeping track of every change for certain types of files is burdensome and can greatly impede on your ability to use git efficiently within a repository. There may also be auxiliary files that you never reference anyway, such as log or output files that are automatically generated. Here is a list of some file types that are generally a good idea to avoid tracking.[2]

- Operating system files: `Thumbs.db`, `.DS_Store`
- Application files: `.Rhistory`, `.Rapp.history`, `.RData`
- Data files: `.xlsx`, `.csv`, `.dta`
- Binary files: `.pdf`, `.docx`, `.pptx`, image files

You can create a file inside your repository to instruct git what kind of files should never be tracked. This file is called `.gitignore`. The period at the beginning means that it is a hidden file (it will not show up in your file viewer unless you have set your options to view hidden files). To create a .gitignore file, navigate to your repository and open a new file with Vim.

[2] See Zell (2015) for a helpful explanation.

```
vim .gitignore
```

In the `.gitignore` file, blank lines and lines starting with `#` are ignored. The asterisk (`*`) is a place holder.

```
# Ignore all Excel files
*.xlsx

# Ignore all files in the directory named Data
Data/

# Track Codebook.xslx even though Excel files are ignored
!Codebook.xlsx
```

There are always exceptions to the above. Perhaps for your project, you want to have a codebook in Excel available to anyone who views the code. If the codebook will not change too much, then it is fine to track it. Another example is that you may want to track the images for the figures needed for your paper. If you use `.jpg` or `.png`, then you can view these images on GitHub itself. Even with these exceptions, it is good practice to maintain a `.gitignore` file, inserting the exceptions or using the force flag (`-f`) as needed.

When starting a project and setting up a repository, you can reference this list to populate your `.gitignore`. Checking `git status` frequently will help you keep the `.gitignore` updated and useful.

### Practice Exercises 1.2

1. On GitHub, create your own (private and empty) repository for your assignments. Call it "Assignments-First-Last" with your first and last names.
2. Clone this repository to the folder you created for this class.
3. From the command line, create a `.gitignore` file. Make it so that your repository will ignore all `.csv` and `.xlsx` files.
4. From the command line, create a file called `README.md`. Write whatever you would like to describe the repository. If in doubt, write: `This repository contains my assingments for the Summer 2022 R and workflow minicourse.`
5. Make it so these changes show up on GitHub (hint: three steps). Use `git status` for guidance.
6. Check that these changes show up on GitHub.
7. Navigate to the repository's settings on GitHub and add me as a collaborator (Settings > Collaborators > Add people). My GitHub username is `aziff`.

## Troubleshooting Git

Generally, there are few things in Git that cannot be undone. But, when you make a mistake or run into trouble, be mindful that the fixes may actually be irreversible. When in doubt, make a defensive copy of your repository before troubleshooting.

### "I committed before making all my changes, but I haven't pushed yet."

You can always make a new commit by staging (`git add`) and committing (`git commit`) the other changes you wanted to make. It is also possible to change the original commit. Make your changes and `git add`. Then, add the `amend` option. This will result in one commit, with the amended commit completely replacing the original commit. This is possible as long as the original commit was not pushed.

```
git commit -m "Initial commit."
git add forgotten-file

# Amend without changing the commit message
git commit --amend --no-edit
```

```
# Amend with an updated commit message
git commit --amend -m "Intial commit proofread for typos."
```

### "I staged a file by accident, but I haven't commited yet."

Suppose you typed `git add *` and accidentally staged a file you did not want to stage. The command `git reset HEAD` will unstage the specified file. The changes made to this file will be saved. Be careful with this command! I suggest against using the `hard` option.

```
git reset HEAD results.txt
git status
```

An alternative approach is to use the below to unstage the specified file. The below does the same using the more current `restore` command.

```
git restore --staged results.txt
git status
```

### "I want to disard my changes to a file since the last commit."

Suppose you change some files the repository and you do not want to keep those changes. In other words, you want to revert some files back to the version in the most recent commit. The below command accomplishes this. This is another potentially dangerous command as it deletes work done locally. Do not use this command unless you are 100% sure you want to delete your changes. If the change was not committed, it will not be saved at all.

```
git checkout -- results.txt
```

An alternative approach is to use `git restore` to revert the file back to the version of the last commit. As for `git checkout`, proceed with caution as local changes will be overwritten.

```
git restore results.txt
```

## Futher Reading

As for any software, there are plentiful resources online, including StackExchange or other forums for specific trouble shooting. The text book Chacon and Straub (2020) is excellent if you want to learn more about the details of Git commands and the underlying system. If you want to see the official help documentation for a command, you can access the manual from the command line by typing `git help <verb>`. For example, this is how you would get the manual on `add`.

```
git help add
```

Sometimes, you just want to check the available options.

```
git add -h
```

There are so many capabilities of Git. The above will help you in this course, but they are just the basics. The textbook Chacon and Straub (2020) goes through all what is possible in Git in an accessible format.

### References


Canonical Ltd. 2021. "The Linux Comand Line for Beginners." *Ubuntu Tutorials*. https://ubuntu.com/tutorials/command-line-for-beginners#1-overview.

Chacon, Scott, and Ben Straub. 2020. *Pro Git: Everything You Need to Know About Git*. 2nd ed. Vol. Version 2.1.277. Apress. https://git-scm.com/book/en/v2.

Matoso, Douglas. 2019. "Using Git (and GitHub) on Windows." *PluralSight*. https://www.pluralsight.com/guides/using-git-and-github-on-windows.
Zell. 2015. "What to Add to Your Gitignore File." https://zellwk.com/blog/gitignore/.