



# Research Reproducibility: Folder Structure and Master Scripts

Roshni Khincha

DIME Analytics – World Bank

February 6, 2020

# Overview

- 1 Introduction
- 2 Folder structure
- 3 Using the DataWork folder
- 4 Master scripts
- 5 Version Control

# Overview

- 1 Introduction
- 2 Folder structure
- 3 Using the DataWork folder
- 4 Master scripts
- 5 Version Control

# Think reproducible

The end goal of this session (and the course in general) is to ensure that all the research we develop is reproducible

Publishing a paper by itself is not enough anymore

- Like tables and figures, code is now an equally important output to share In this context, publishing code is pointless if it is
  - ▶ not reproducible, (i.e. not getting the same results) and
  - ▶ no one understands how to run it
- For code to be useful, it requires transparency, accountability, and an easy to understand workflow
- Basically, code needs to be organized and readable

## This is a lot easier said than done

- At DIME, we have large teams collaborating on the same codes and data sets
- Long projects easily become complex as they have multiple rounds/sources of data which need to be organized
- Standardizing organization of documents and code prevents mistakes and reduces the cost of transitioning across projects and teams

# What do we mean by data management?

In this session we will understand and implement best practices to manage data work through

- Setting up a good folder structure
- Creating a master script which runs all code
- Establishing a version control system

When the contents of this session are applied to a project, anyone with full access to its files and folders should be able to replicate the research and understand the structure of the data work

# Overview

- 1 Introduction
- 2 Folder structure**
- 3 Using the DataWork folder
- 4 Master scripts
- 5 Version Control

## Why should you care about folder structure?

- Your project folder probably has a lot of subfolders for literature, presentations, concept notes, and other documents
- On this session, we will focus on the folders relate to data work. We will call the set of data-related folders the DataWork folder
- The `ietoolkit` Stata package offers a command called `iefolder` that helps set up the folder structure and its interactions with code files
- *iefolder* provides a template for the folder structure for a typical DIME project using primary data
- We will not get into the details of how to use the `iefolder` command here, but rather focus on the principles behind it
- For documentation and details on how to use the command, type `help iefolder` in Stata

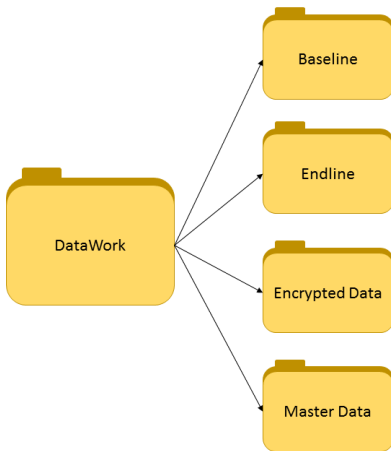


## Why should you care about folder structure?

- The motivation behind `iefolder` is to create a standardized structure that is easy for team members to navigate
- Different projects may have specific needs, but templates used in *iefolder* are a good starting point to think through the folder structure for any project
- Whatever structure you are using, however, applying it to all your projects will make it easier to move across projects

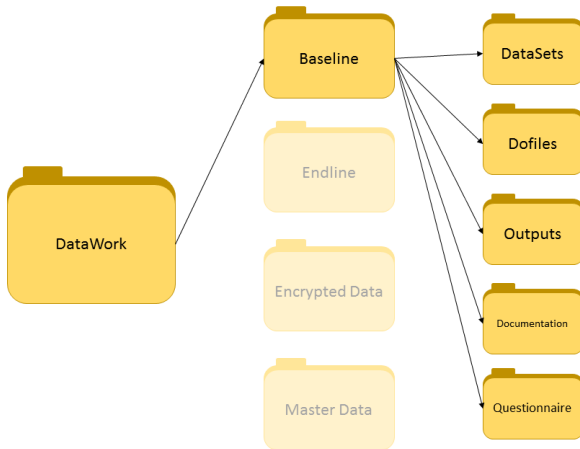
## DataWork Folder: Overview

This is what the DataWork folder created by iefolder looks like:



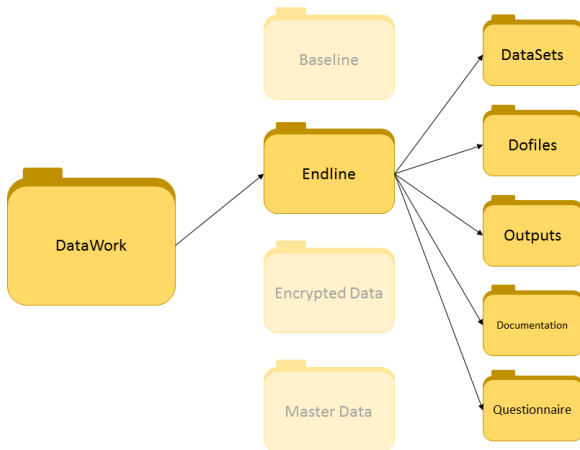
## DataWork Folder: Rounds folders

The baseline folder will store all baseline data, as well as do-files and outputs that refer exclusively to this round of data collection.



## DataWork Folder: Rounds folders

The endline folder will store all endline data, as well as code and outputs that refer exclusively to this round of data collection. Note that its structure is exactly the same as the structure of the baseline folder.

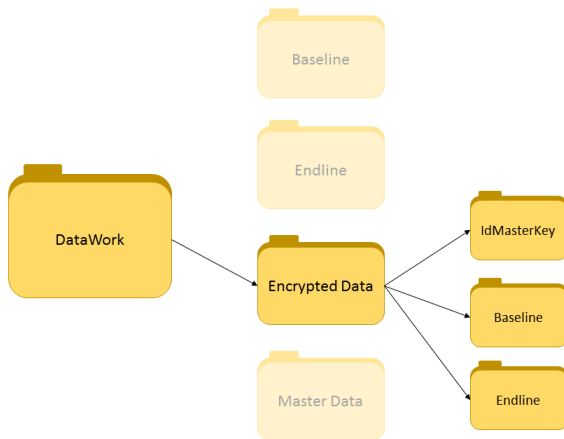


## DataWork Folder: Rounds folders

- While a “round” of data collection may only seem applicable to primary data, you can think of a “round” as one *source* of data collection
- Another way to put it is to think of a “round” as a set of data that will be processed using the same code
- One example of this could be a primary data collection that includes two levels of observation (household and community, student and school, patient and doctor, etc): each level will probably have its own questionnaire, and will be cleaned separately. So you will create different round folders, say `HouseholdBaseline` and `CommunityBaseline`.
- Another example is when the same questionnaire is applied twice, but the variable names and value labels are slightly different. Then you will also need two separate “round” folders

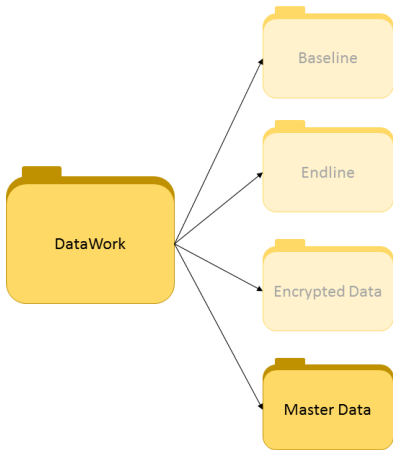
## DataWork Folder: Encrypted Data

The encrypted data folder will contain all personally identifiable data for each round of data collection, and a folder with ID master keys linking each unidentified ID to the identified observations. As the name suggests, this folder should be encrypted.



## DataWork Folder: Master Data

The master data folder will store the master data sets for each unit of observation in your project.



# What is a master data set?

- A comprehensive listing or comprehensive listing of all potential observations that one can encounter during the course of a project
- It forms a comprehensive documentation of actions taken for each unit of observation in the scope of the project
- An example: A household master data set will include
  - ▶ households listed in the census
  - ▶ households sampled for the surveys
  - ▶ households included in monitoring (even if not part of the project)
  - ▶ households included in analysis
  - ▶ an unique ID for each of these household

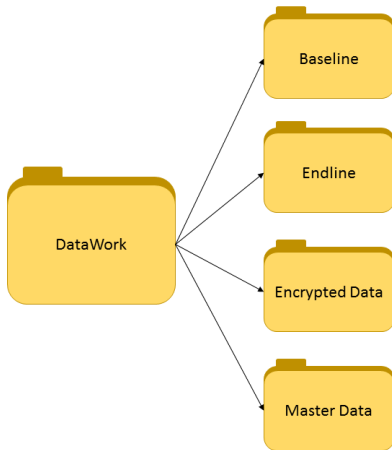


# Overview

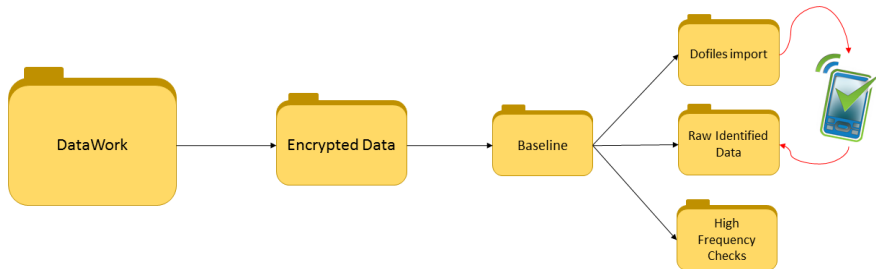
- 1 Introduction
- 2 Folder structure
- 3 Using the DataWork folder**
- 4 Master scripts
- 5 Version Control

# Using the DataWork Folder

So you received data from the field. What now?



## Using the DataWork Folder: EncryptedData

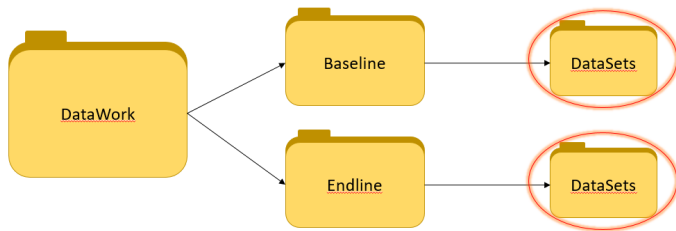


- The raw data with identifying information should be stored in the EncryptedData folder

## Using the DataWork Folder: EncryptedData

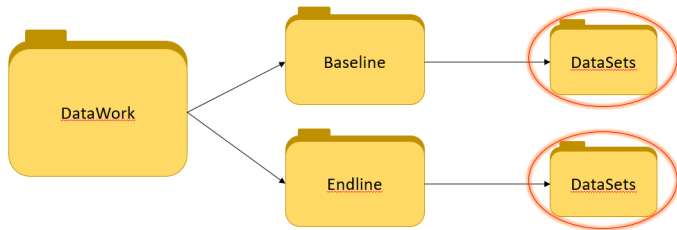
- The identified data **must be safely stored**. This may mean different things in different contexts
- The first thing that you should do when receiving data is to de-identify it so it can be safely used and shared. We will discuss how to do this in our next session
- Only team members authorized in the IRB should be able to access the identified data
- One way to make sure that identifying data is securely stored is to only keep an encrypted version of it in any shared folder, and share de encryption key separately
- How to implement this in practice is beyond the scope of this session
- For more information on what secure storage is and how to implement it, visit the DIME Wiki

## Using the DataWork Folder: DataSets



- All data in the rounds folders should be de-identified!

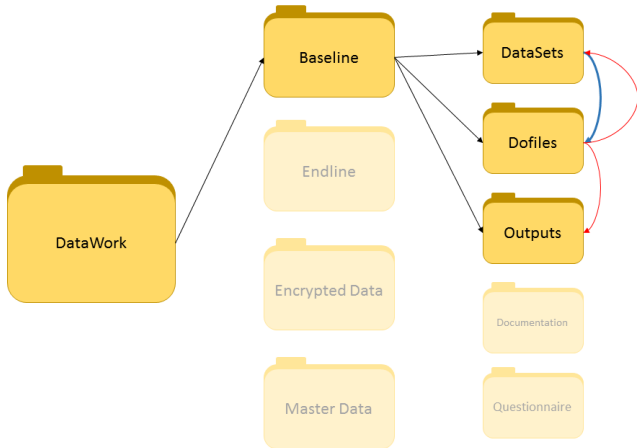
## Using the DataWork Folder: DataSets



- All data in the rounds folders should be de-identified
- If you don't know how to de-identify your data, you can find instructions for that in DIMEwiki

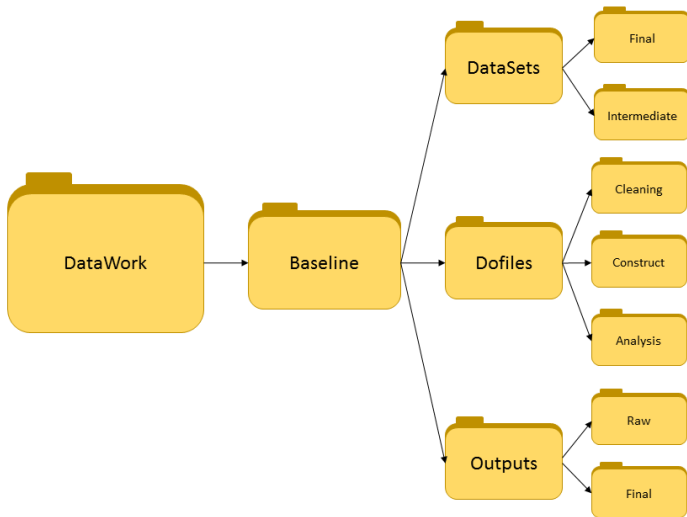
## Using the DataWork Folder: Round folders

The scripts in each round folder will load data from that round's DataSets folder and store any outputs in the round's Outputs folder



## Using the DataWork Folder: Rounds folders

This is what one specific round folder looks like in detail

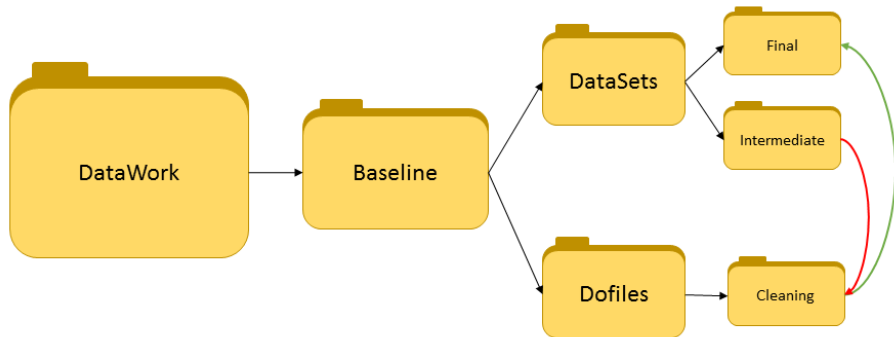




## Using the DataWork Folder: Cleaning

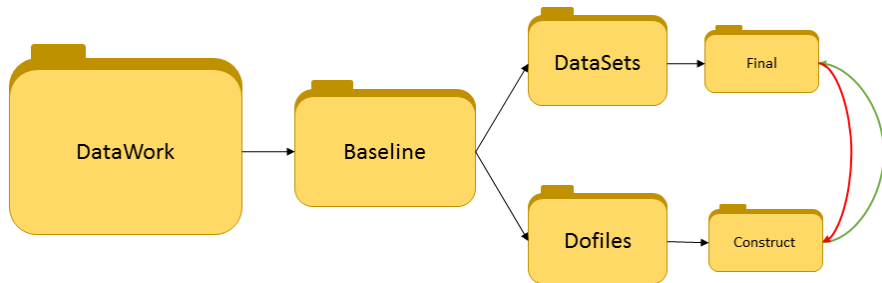
The codes in the Cleaning folder will

- 1 Load the intermediate data sets (e.g., de-identified survey data)
- 2 Clean them (correct, annotate, label, recode)
- 3 Save the clean data sets in the Final DataSets folder
- 4 We will cover the specifics of Data Cleaning in more detail later on in the course



## Using the DataWork Folder: Construct

- The scripts in the Construct folder load the clean data and use it to construct the indicators that will be used for analysis. This is when new variables are created
- The constructed data set, containing these indicators, will be stored in the Final DataSets folder
- We will cover the specifics of Data Construction in more detail later on in the course

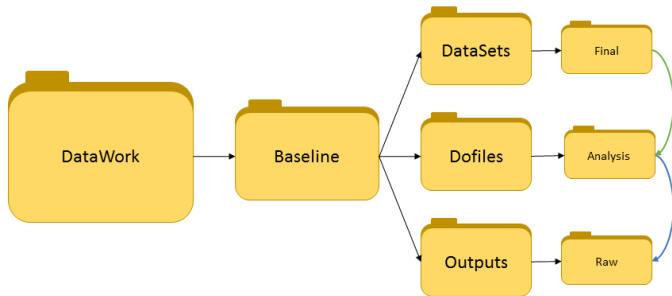


## Using the DataWork Folder: Round folders

- Analysis scripts should never create new variables
- All data modification should be made in the Construct scripts
- If you need a variable for analysis that is not in the data set you loaded, go back to the construct do-file and add it

## Using the DataWork Folder: Analysis

- The scripts in the Analysis Dofiles folder load the constructed data and run the analysis
- Outputs such as plots and tables generated by these scripts are stored in the Raw Outputs folder
- **Do not** create variables in the analysis scripts!



## Using the DataWork Folder: Analysis

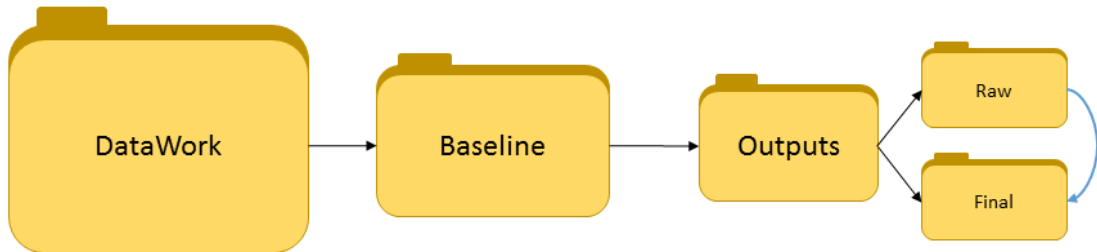
To emphasize

- Analysis do-files should never create new variables
- All data modification should be made in the Construct do-files
- If you need a variable for analysis that is not in the data set you loaded, go back to the construct do-file and add it

We will cover the specifics of Data Analysis in more detail later on in the course

## Using the DataWork Folder: Outputs

- The Final Outputs folder stores reports, papers and other documents you create using the files in the Raw Outputs folder



# Overview

- 1 Introduction
- 2 Folder structure
- 3 Using the DataWork folder
- 4 Master scripts**
- 5 Version Control

## What's the need for a master script?

- As you might have noticed, we mentioned the creation of many code scripts in the previous slides
- A big project can become very complex, and scripts need to be run in a certain order to create the right output
- That could mean you'd need to write one extremely long script, or a different document with instructions about in which order to run all the scripts
- **However**, you can make a script that runs other scripts
- This makes it easy for anyone reproducing your code to do it with ease



## What is a master script?

- The master script is the map over all data work in your data folder
- It is the table of content for the instructions that you code
- It should be possible to follow all data work in the data folder, from raw data to analysis output, by reading the master script

# What is a master script?

```
/*=====
 *                               TEMPLATE MASTER DO-FILE
 *                               DIME 2020
 *=====
PART 1: Set standard settings and install packages
/*=====

if (0) {
  %ssc install ietoolkit, replace
}

ieboilstart, v(15.1)
`r(version)'

/*=====
PART 2: Prepare folder paths and define programs
/*=====

* Research Assistant folder paths
if "`c(username)'" == "ResearchAssistant" {
  global github      "C:/Users/RA/Documents/GitHub/d4di/DataWork"
  global dropbox      "C:/Users/RA/Dropbox/d4di/DataWork"
  global encrypted    "A:/DataWork/EncryptedData" // Always mount to A disk!
}

* Baseline folder globals
global bl_encrypt     "${encrypted}/Round Baseline Encrypted"
global bl_dt          "${dropbox}/Baseline/DataSets"
global bl_ds          "${dropbox}/Baseline/Documentation"
global bl_do          "${github}/Baseline/Dofiles"
global bl_out         "${github}/Baseline/Output"

/*=====
PART 3: Run do files
/*=====

/*-----
PART 3.1: De-identify baseline data
/*-----

REQUIRES:  $(bl_encrypt)/Raw Identified Data/D4DI_baseline_raw_identified.dta
CREATES:   $(bl_dt)/Raw Deidentified/D4DI_baseline_raw_deidentified.dta
IDS VAR:   hhid

do "$(bl_do)/Cleaning/deidentify.do"

}

/*-----
PART 3.3: Construct income indicators
/*-----

REQUIRES:  $(bl_dt)/Final/D4DI_baseline_clean.dta
CREATES:   $(bl_out)/Raw/D4DI_baseline_income_distribution.png
           $(bl_dt)/Intermediate/D4DI_baseline_constructed_income.dta
IDS VAR:   hhid

do "$(bl_do)/Construct/construct_income.do"
```

```
/*=====
 *                               ML2 PROJIRRI
 *                               R MASTER
 *                               2017
 *=====
# Select actions -----

install_packages <- 0
clean_nhanandenbe <- 0 # Cleans data for the Nhanandenbe scheme
clean_campo4 <- 0 # Cleans data for the Campo 4 scheme

# User paths -----

# Luiza's bank computer
if (Sys.getenv("USERNAME") == "wb501238") {
  github <- file.path("C:/Users/wb501238/Documents/GitHub/PROJIRRI-Pilot/DataWork",
    fsep = .Platform$file.sep)
  dropbox <- file.path("C:/Users/wb501238/Dropbox/WB/PROJIRRI Survey/Measurement pilot/Data collections",
    fsep = .Platform$file.sep)
}

# Luiza's personal computer
if (Sys.getenv("USERNAME") == "Larissa") {
  github <- file.path("D:/Usurrio/Documents/GitHub/PROJIRRI-Pilot/DataWork",
    fsep = .Platform$file.sep)
  dropbox <- file.path("D:/Usurrio/Dropbox/WB/PROJIRRI Survey/Measurement pilot/Data collections",
    fsep = .Platform$file.sep)
}

# Packages used -----

packages <- c("readstata13",
  "tidyr",
  "plyr",
  "devtools")

# Install packages that are not yet installed
supply(packages, function(x) {
  if (!x %in% installed.packages()) {
    install.packages(x, dependencies = TRUE)
  }
})

# Load all packages -- this is equivalent to using library(package) for each
# package listed before
invisible(supply(packages, require, character.only = TRUE))

# Folder globals -----

data_raw <- file.path(dropbox, "data", "Irrigation water data", "Raw")
data_int <- file.path(dropbox, "data", "Irrigation water data", "Intermediate")
data_final <- file.path(dropbox, "data", "Irrigation water data", "Final")
code <- file.path(github, "do-files", "Irrigation water data")

# Run code -----

if (clean_nhanandenbe) {source(file.path(code, "Nhanandenbe.R"))}
if (clean_campo4) {source(file.path(code, "Campo 4.R"))}
```

## Master script: allows for easy collaboration

- If we share a project over DropBox or GitHub, all team members have the same folder structure
- A master script allows multiple people to set their own global to the project folder
- This way, anyone sharing the project folder can easily run your scripts

```
*Set this value to the user currently using this file
global user 1 // Luiza
global user 2 // Ben

* Root folder globals
* -----

if $user == 1 {
    global projectfolder "C:\Users\WB501238\Dropbox\DIME\RA survey"
}

if $user == 2 {
    global projectfolder "C:\Users\Benjamin\Dropbox\Work\DIME\RA survey"
}
```

Master script: allows for easy collaboration

- If we share a project over DropBox or GitHub, all team members have the same folder structure
- A master script allows multiple people to set their own global to the project folder
- This way, anyone sharing the project folder can easily run your scripts

```
# User paths -----
# Luiza's bank computer
if (Sys.getenv("USERNAME") == "wb501238") {
  github    <- file.path("C:/Users/WB501238/Documents/GitHub/PROIRRI-Pilot/DataWork",
                        fsep = .Platform$file.sep)
  dropbox   <- file.path("C:/Users/WB501238/Dropbox/WB/PROIRRI Survey/Measurement pilot/Data collections",
                        fsep = .Platform$file.sep)
}

# Luiza's personal computer
if (Sys.getenv("USERNAME") == "Larissa") {
  github    <- file.path("D:/Usuário/Documents/GitHub/PROIRRI-Pilot/DataWork",
                        fsep = .Platform$file.sep)
  dropbox   <- file.path("D:/Usuário/Dropbox/WB/PROIRRI Survey/Measurement pilot/Data collections",
                        fsep = .Platform$file.sep)
}
```

## Master script: connection between code and folder structure

- The master script contains globals or objects referencing the subfolder in the DataWork folder, so you have easy shortcuts to them
- Any change in folder structure can be easily accounted for by changing the folder global
- `iefolder` automates this process by creating folders and updating the master at the same time

# Master script: connection between code and folder structure

```
* Project folder globals
* -----

global dataWorkFolder      "$projectfolder/DataWork"

*iefolder*1*FolderGlobals*master*****
*iefolder will not work properly if the line above is edited

global mastData            "$dataWorkFolder/MasterData"

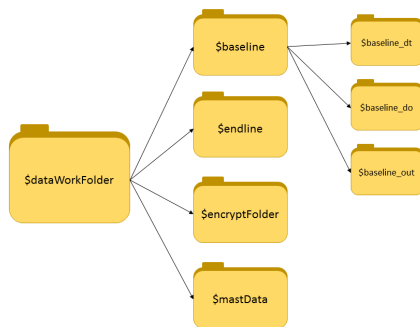
*iefolder*1*FolderGlobals*rawData*****
*iefolder will not work properly if the line above is edited

global encryptFolder       "$dataWorkFolder/EncryptedData"
global masterIdDataSets    "$encryptFolder/IDMasterKey"

*iefolder*1*RoundGlobals*rounds*baseline*****
*iefolder will not work properly if the line above is edited

*baseline folder globals
global baseline            "$dataWorkFolder/baseline"
global baseline_dt        "$baseline/DataSets"
global baseline_do        "$baseline/Dofiles"
global baseline_out       "$baseline/Output"
```

(a) Master do-file



(b) Folder structure

## Master script: allows easy updates

- User inputs and global settings should all be defined in the master script
- Examples of this include folder paths, conversion rates, control and outcome variables, and even graph colors
- This allows you to make changes to a single line of code through a global or object whenever you want to apply an update to all your codes
- If you are using Stata, globals must *only* be defined in the master do file

Master script: allows easy updates

[illegible]






## Master script: allows for easy replication

- Anyone should be able to follow and to reproduce all your work from raw data to all outputs with one click in this script, after adding only their root folder path
- In general, you should always run codes through the master script
- This prevents the very common workflow of “run this script, then this one, and finally that other one”
- It also helps you make sure that changes you make to one piece of code don't break other codes

## Master script: the map to all data work

- By reading the master script, someone external to the project should have a general understanding of what is being done at every step
- If you want to see how a particular table or data set or created, reading the master script should be enough to tell which script to look at
- Using locals and objects to create “switches” to select which parts of the project to run or not facilitate the usage of the code when projects are very long and complex

## Master script: the map to all data work

```
# Select actions -----  
  
clean_nhamandembe <- 0 # Cleans data for the Nhamandembe scheme  
clean_campo4      <- 0 # Cleans data for the Campo 4 scheme  
  
# User paths   
# Packages used   
# Folder globals   
# Run code -----  
  
if (clean_nhamandembe) {source(file.path(code,"Nhamandembe.R"))}  
if (clean_campo4)      {source(file.path(code,"Campo 4.R"))}
```

## Master script: the map to all data work

```
* Main figures =====

*****
* Take-up of women's car by opportunity cost                                *
*-----*
*   REQUIRES: ${dt_rider_fin}/pooled_rider_audit_constructed.dta          *
*   CREATES:  ${out_graphs}/takeup_fe.png                                *
*             ${out_graphs}/takeup_person.png                            *
*****

do "${do_analysis}/Rider audits/Plots/takeup.do"

*****
* IAT D-Score distribution by instrument and gender                        *
*-----*
*   REQUIRES: ${dt_platform_fin}/platform_survey_constructed.dta          *
*   CREATES:  ${out_graphs}/IAT_safety.png                                *
*             ${out_graphs}/IAT_advances.png                              *
*             ${out_graphs}/IAT_men.png                                   *
*             ${out_graphs}/IAT_women.png                                 *
*****

do "${do_analysis}/Platform survey/Plots/iatscores.do"
```

# Overview

- 1 Introduction
- 2 Folder structure
- 3 Using the DataWork folder
- 4 Master scripts
- 5 Version Control**

# Version Control

- Version control is a crucial part of data management as
  - ▶ code and documents undergo a lot of revision and redrafting
  - ▶ multiple users make changes to code and documents
- It is essential to set up version control at the start of the project to avoid glitches in later stages
- There are a few ways of implementing version control
  - ▶ Through setting up file naming conventions
  - ▶ Using a syncing software such as Dropbox, OneDrive, etc.
  - ▶ Using git/GitHub

## How to version control?

- Using file naming conventions (such as adding dates and initials as suffixes) is better than no version control, but it can get very unwieldy very quickly
- Syncing softwares allow teams to revert to old version of a document but doesn't allow to keep track of specific changes made
- git is currently the best version control system out there as one can track changes and revert to old versions easily

## Recommended practices for version control

- DIME Analytics recommends using git for version control and all DIME projects are required to use git for version control
- Anything that can be version controlled through git should be version controlled (meaning all plain text files, including outputs)
- There's a steep learning curve for git/GitHub and will be covered in greater detail later on in the course



Questions?

## Recommended practices for file paths

- \* Dynamic, absolute file paths

- \* Dynamic (and absolute) - GOOD

- ```
global myDocs      "C:/Users/username/Documents"
global myProject   "${myDocs}/MyProject"
use "${myProject}/MyDataset.dta"
```

- \* Relative and absolute file paths

- \* Relative - BAD

- ```
cd "C:/Users/username/Documents/MyProject"
use MyDataset.dta
```

- \* Absolute but not dynamic - BAD

- ```
use "C:/Users/username/Documents/MyProject/MyDataset.dta"
```