# Best Practices for High-Quality Data Science

**Scott Teresi**

**Department of Horticulture**

May 21, 2024

MICHIGAN STATE
UNIVERSITY

# Objective:

1. Communicate the importance of reproducibility and good coding practices
2. Provide examples of practices to avoid
3. Raise awareness of tools and approaches that can help you be a better scientist

# My Background:

- Graduated from The College of William & Mary in 2019
- I majored in biology and minored in computer science

# My Background:

- Graduated from The College of William & Mary in 2019
- I majored in biology and minored in computer science
- I was part of the REU in 2018 in Pat Edger's lab

# My Background:

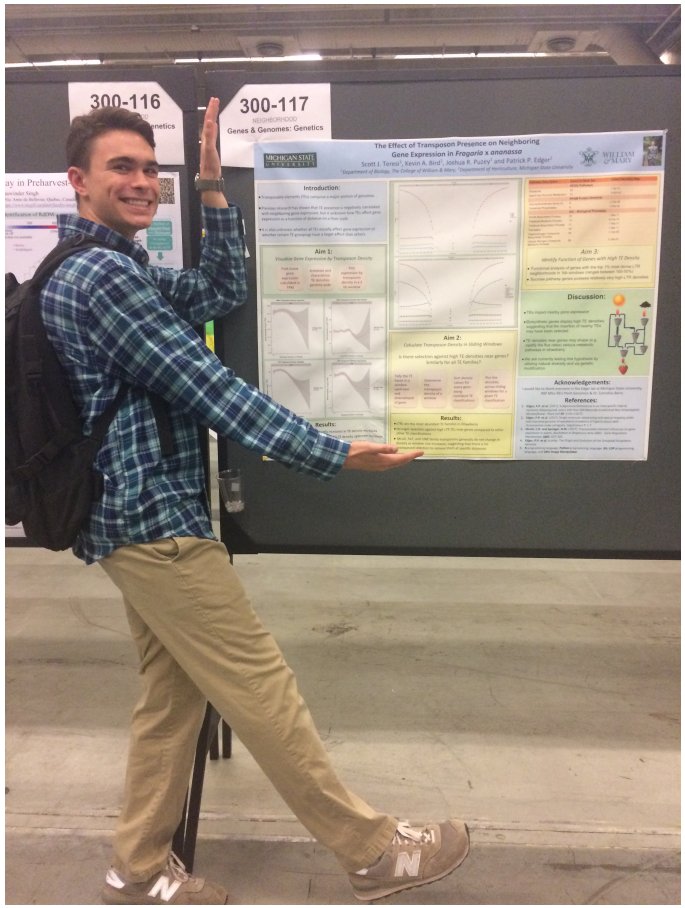
- Graduated from The College of William & Mary in 2019
- I majored in biology and minored in computer science
- I was part of the REU in 2018 in Pat Edger's lab
- Currently a PhD student in the Dept. of Horticulture and in Genetics & Genome Sciences program

# My Background:

- Graduated from The College of William & Mary in 2019
- I majored in biology and minored in computer science
- I was part of the REU in 2018 in Pat Edger's lab
- Currently a PhD student in the Dept. of Horticulture and in Genetics & Genome Sciences program
- I wrote code to fill a knowledge gap in the transposable element community

# My Background:

- Graduated from The College of William & Mary in 2019
- I majored in biology and minored in computer science
- I was part of the REU in 2018 in Pat Edger's lab
- Currently a PhD student in the Dept. of Horticulture and in Genetics & Genome Sciences program
- I wrote code to fill a knowledge gap in the transposable element community
- The code showed some really promising preliminary results in strawberry, but it was not publishable or easy to run on additional genomes.



# Outline

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration
- Essential to saving you time & avoiding future headaches

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration
- Essential to saving you time & avoiding future headaches
- Essential to minimizing wasted effort

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration
- Essential to saving you time & avoiding future headaches
- Essential to minimizing wasted effort
- Essential to being professional, and having you and your work taken seriously

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration
- Essential to saving you time & avoiding future headaches
- Essential to minimizing wasted effort
- Essential to being professional, and having you and your work taken seriously
  - Your code is a product and a reflection on you!

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration
- Essential to saving you time & avoiding future headaches
- Essential to minimizing wasted effort
- Essential to being professional, and having you and your work taken seriously
  - Your code is a product and a reflection on you!
- Essential to getting a job!

# Data Reproducibility & Code Quality is:

- Essential to the forward progress of science
- Essential to communication and collaboration
- Essential to saving you time & avoiding future headaches
- Essential to minimizing wasted effort
- Essential to being professional, and having you and your work taken seriously
  - Your code is a product and a reflection on you!
- Essential to getting a job!
- Essential to getting published!

# What is an Anti-Pattern?

"An anti-pattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences."

# What is an Anti-Pattern?

"An anti-pattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences."

- Anti-Patterns are a method for efficiently mapping a general situation to a specific class of solutions; they provide a common vocabulary for identifying problems and discussing solutions.

# What is an Anti-Pattern?

"An anti-pattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences."

- Anti-Patterns are a method for efficiently mapping a general situation to a specific class of solutions; they provide a common vocabulary for identifying problems and discussing solutions.
- The concept of anti-patterns originated in the software engineering community, but they are applicable to many fields. It is a riff on the book "Design Patterns: Elements of Reusable Object-Oriented Software" by the Gang of Four.

# What is an Anti-Pattern?

"An anti-pattern is a literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences."

- Anti-Patterns are a method for efficiently mapping a general situation to a specific class of solutions; they provide a common vocabulary for identifying problems and discussing solutions.
- The concept of anti-patterns originated in the software engineering community, but they are applicable to many fields. It is a riff on the book "Design Patterns: Elements of Reusable Object-Oriented Software" by the Gang of Four.
- Sometimes it is easier to identify what not to do, rather than what to do.

# Root Causes of Bad Code:

- Haste
  - "*It works though!*" or frequently "*Just make it work!*"

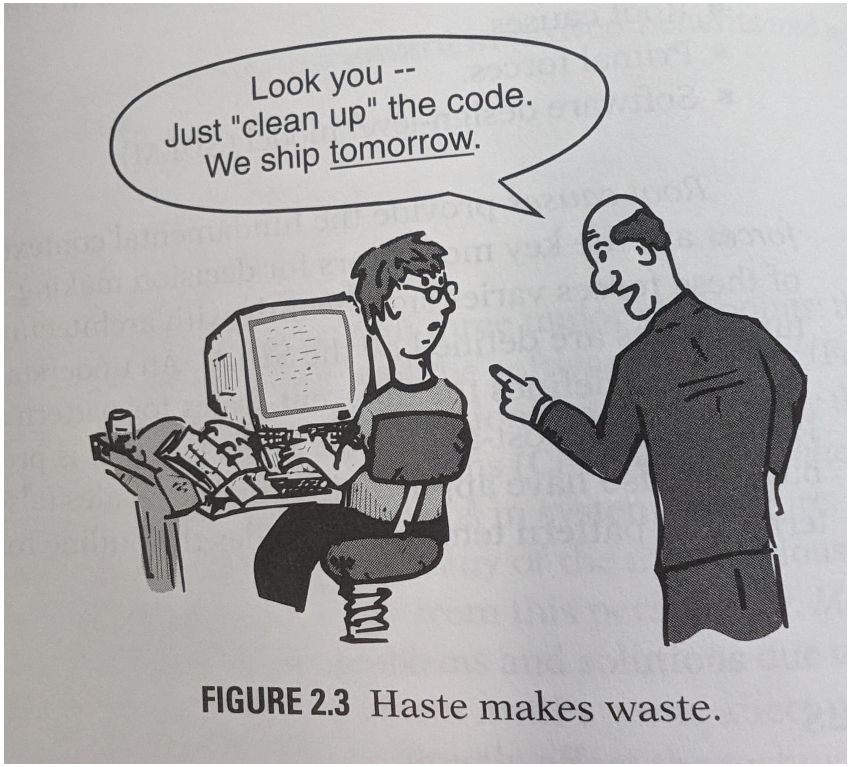



**FIGURE 2.3** Haste makes waste.

# Root Causes of Bad Code:

- Apathy
  - You will inevitably need to come back to your code at some point.

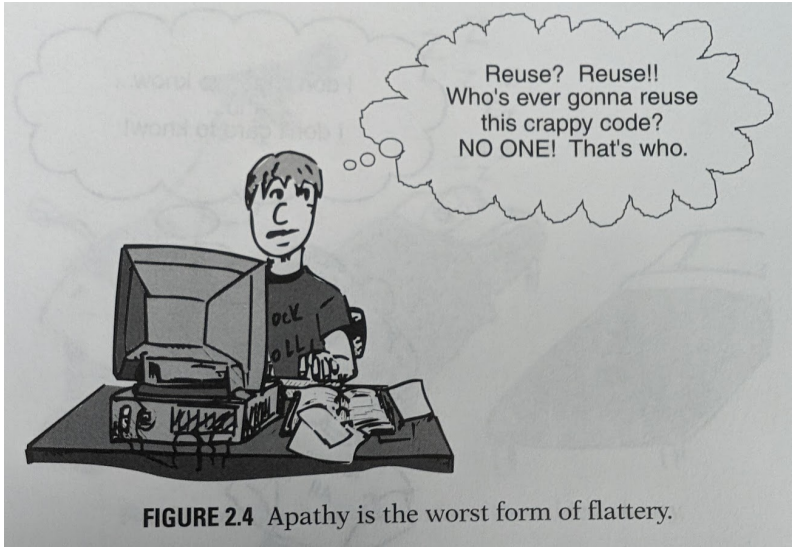



**FIGURE 2.4** Apathy is the worst form of flattery.

# Root Causes of Bad Code:

- Narrow-Mindedness
  - Not reading the manual
  - Not reading the manual
  - Not reading the manual

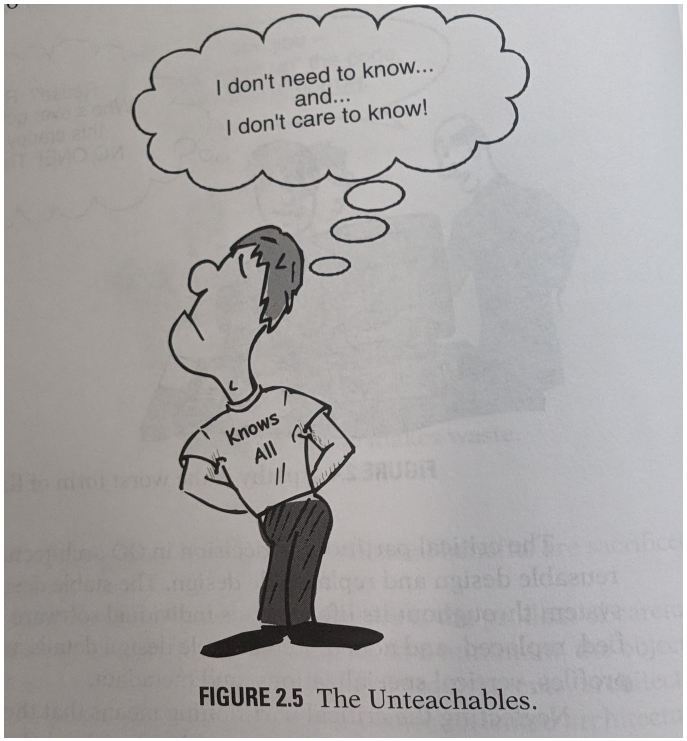
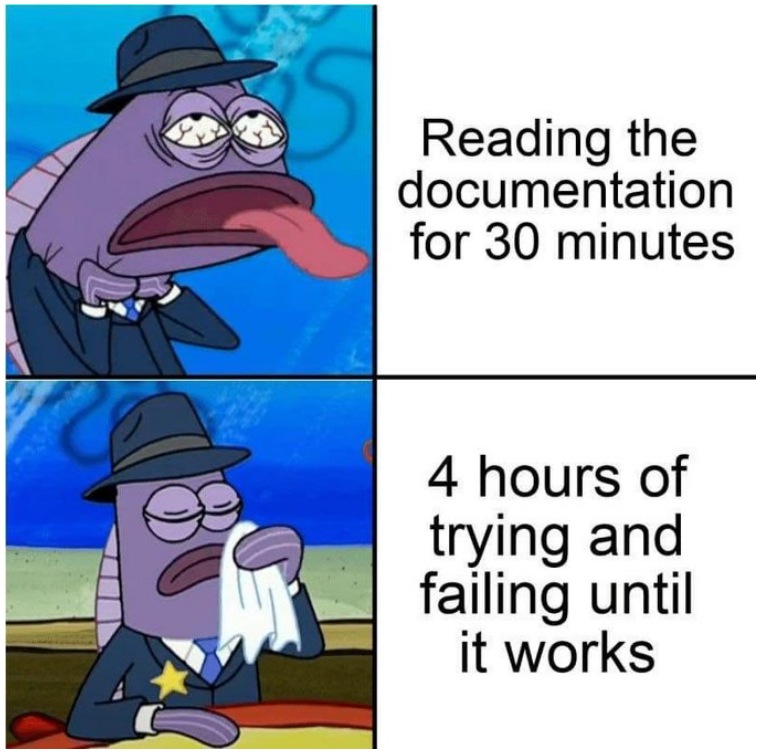


FIGURE 2.5 The Unteachables.

# Root Causes of Bad Code:

- Narrow-Mindedness
  - Not reading the manual
  - Not reading the manual
  - Not reading the manual



# Root Causes of Bad Code:

- Avarice
  - Overly complex systems are difficult to develop, test, document, maintain, and extend.
  - You are not paid by the line of code, and you aren't cool for writing complex code.
  - Complexity *very, very* bad — `https://grugbrain.dev/`

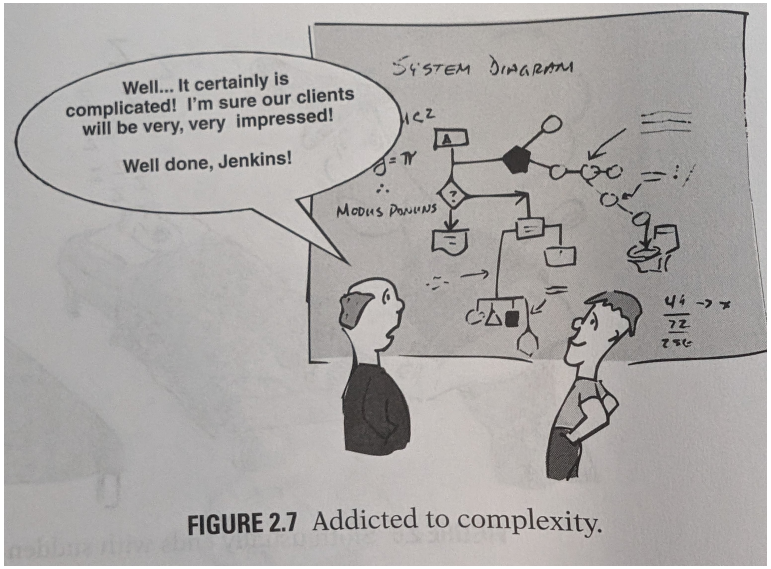
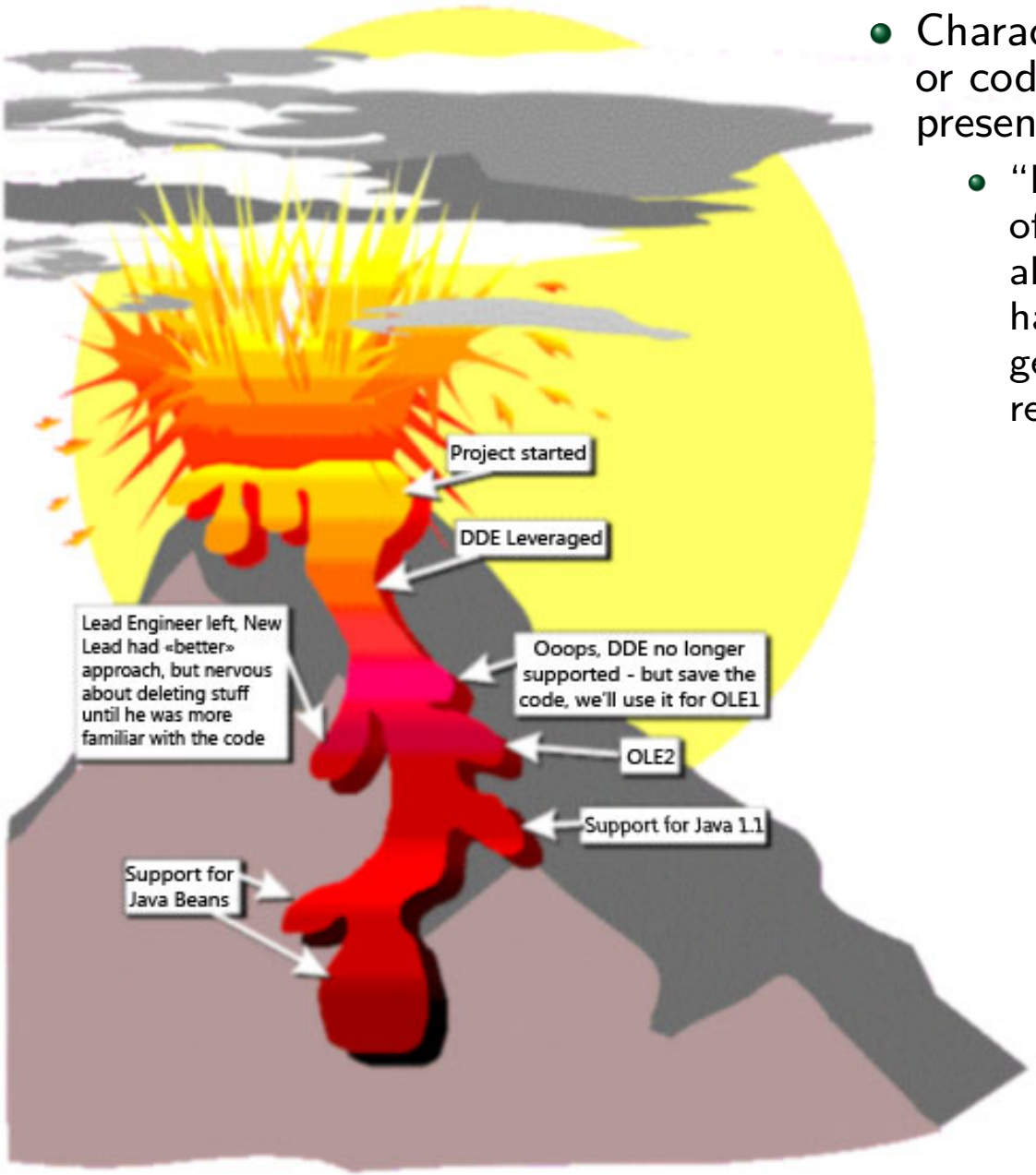



**FIGURE 2.7** Addicted to complexity.

# The Blob:

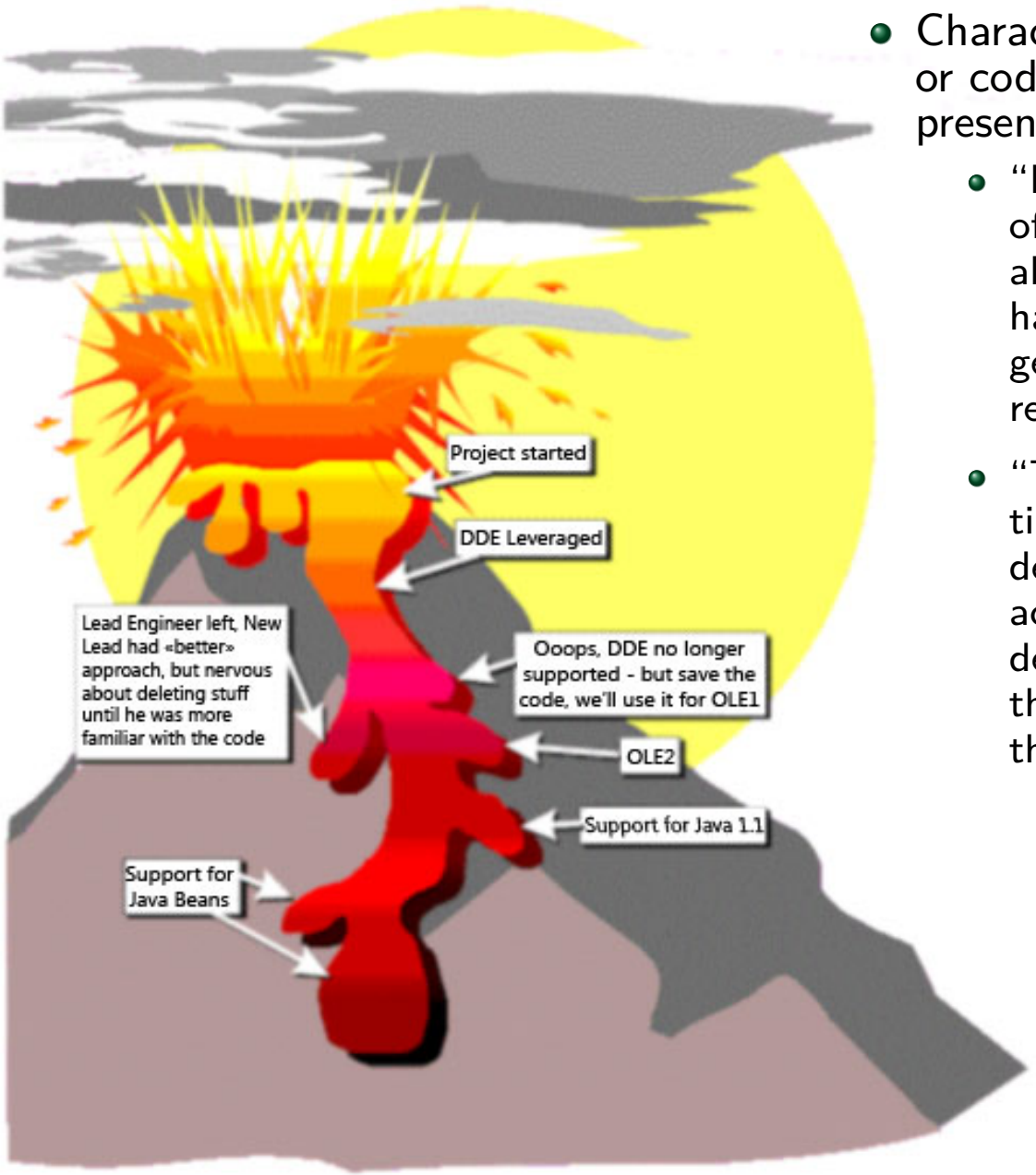
- **The Blob:** Characterized by a lack of architecture (what does this mean?), and a lack of separation of concerns & responsibility.
    - The blob in bioinformatics is often a singular script that does everything and it has few functions; everything happens in one unstoppable, untestable, procedure. The blob heavily minimizes code reuse.
    - Modifications to components (if there are any!) of the blob can have unintended consequences, as the blob is a tightly coupled system.

# Lava Flow:



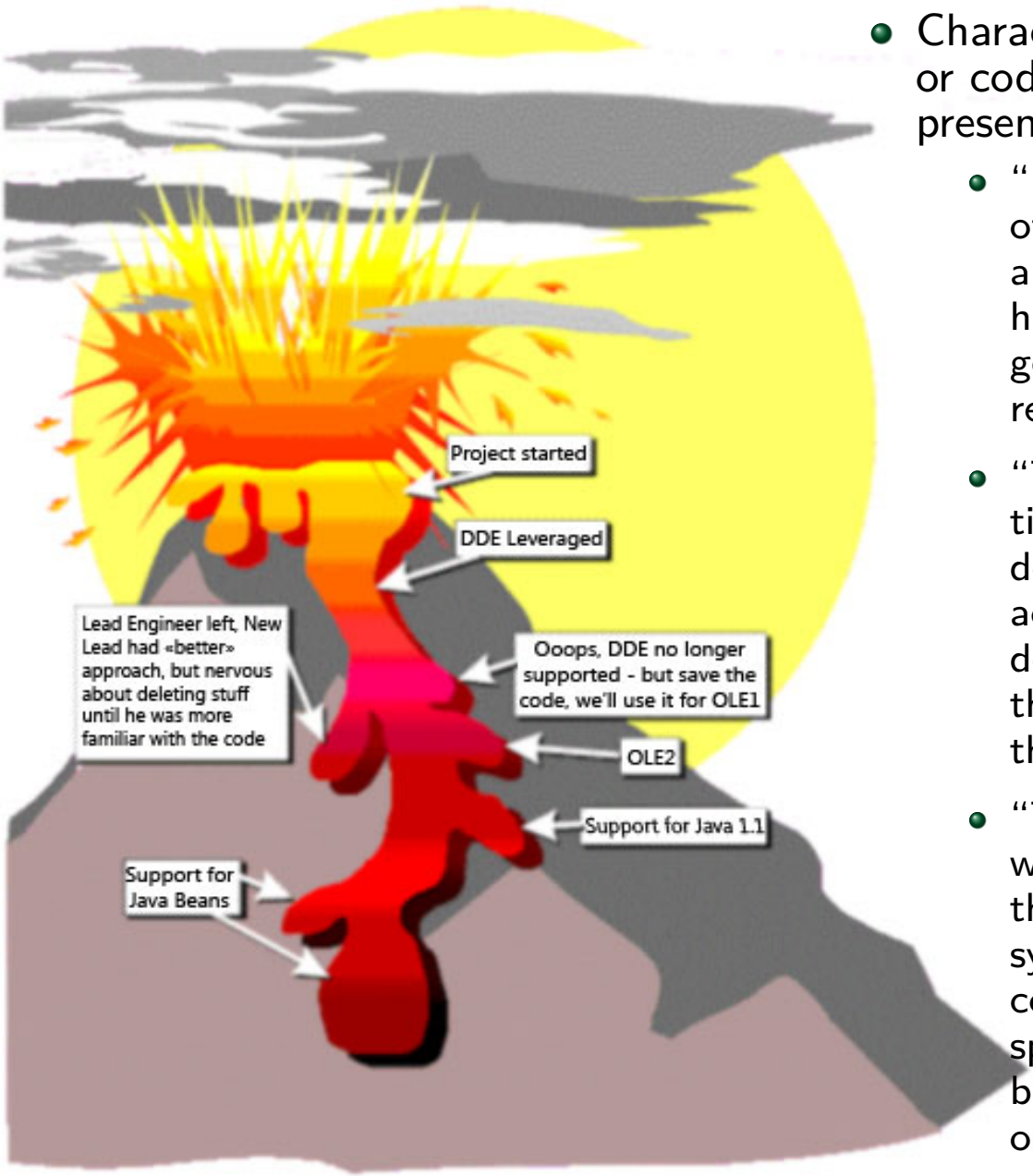

- Characterized by the presence of dead code, or code that is no longer used, but is still present in the codebase.
  - "It is characterized by the lavalike "flows" of previous developmental versions strewn about the code landscape, which have now hardened into a basaltlike, immovable, generally useless mass of code no one can remember much, if anything about."

# Lava Flow:




- Characterized by the presence of dead code, or code that is no longer used, but is still present in the codebase.
  - "It is characterized by the lavalike "flows" of previous developmental versions strewn about the code landscape, which have now hardened into a basaltlike, immovable, generally useless mass of code no one can remember much, if anything about."
  - "This is the result of earlier developmental times, when, while in a research mode, developers tried out several ways of accomplishing things, typically in a rush to deliver some kind of demonstration, thereby casting sound design practices to the winds and sacrificing documentation"

# Lava Flow:




- Characterized by the presence of dead code, or code that is no longer used, but is still present in the codebase.
  - "It is characterized by the lavalike "flows" of previous developmental versions strewn about the code landscape, which have now hardened into a basaltlike, immovable, generally useless mass of code no one can remember much, if anything about."
  - "This is the result of earlier developmental times, when, while in a research mode, developers tried out several ways of accomplishing things, typically in a rush to deliver some kind of demonstration, thereby casting sound design practices to the winds and sacrificing documentation"
  - "The result is several fragments of code, wayward variable classes, and procedures that are not clearly related to the overall system. In fact, these flows are often so complicated in appearance and spaghettilike that they seem important, but no one can really explain what they do or why they exist."

# Spaghetti Code:

- **Spaghetti Code:** Very little organization, and no separation of concerns. This is most often seen alongside the blob anti-pattern, and is really common for beginners. Code often resembles a single, multistage process flow, and the flow of execution is dictated by the implementation of the code, rather than the design of the code.

# Spaghetti Code:

- **Spaghetti Code:** Very little organization, and no separation of concerns. This is most often seen alongside the blob anti-pattern, and is really common for beginners. Code often resembles a single, multistage process flow, and the flow of execution is dictated by the implementation of the code, rather than the design of the code.

- **How can you address it?** Refactoring your code at regular intervals, do not wait until the end! Later, when we talk about Git, I will show you how to use branches to develop new features or fix bugs; when you are ready to finalize a branch, that is a good time to do some clean-up.

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning
- Write your entire project in one script

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning
- Write your entire project in one script
- Have zero Separation of Concerns

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning
- Write your entire project in one script
- Have zero Separation of Concerns
- Make your code complex

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning
- Write your entire project in one script
- Have zero Separation of Concerns
- Make your code complex
- Lack of version control

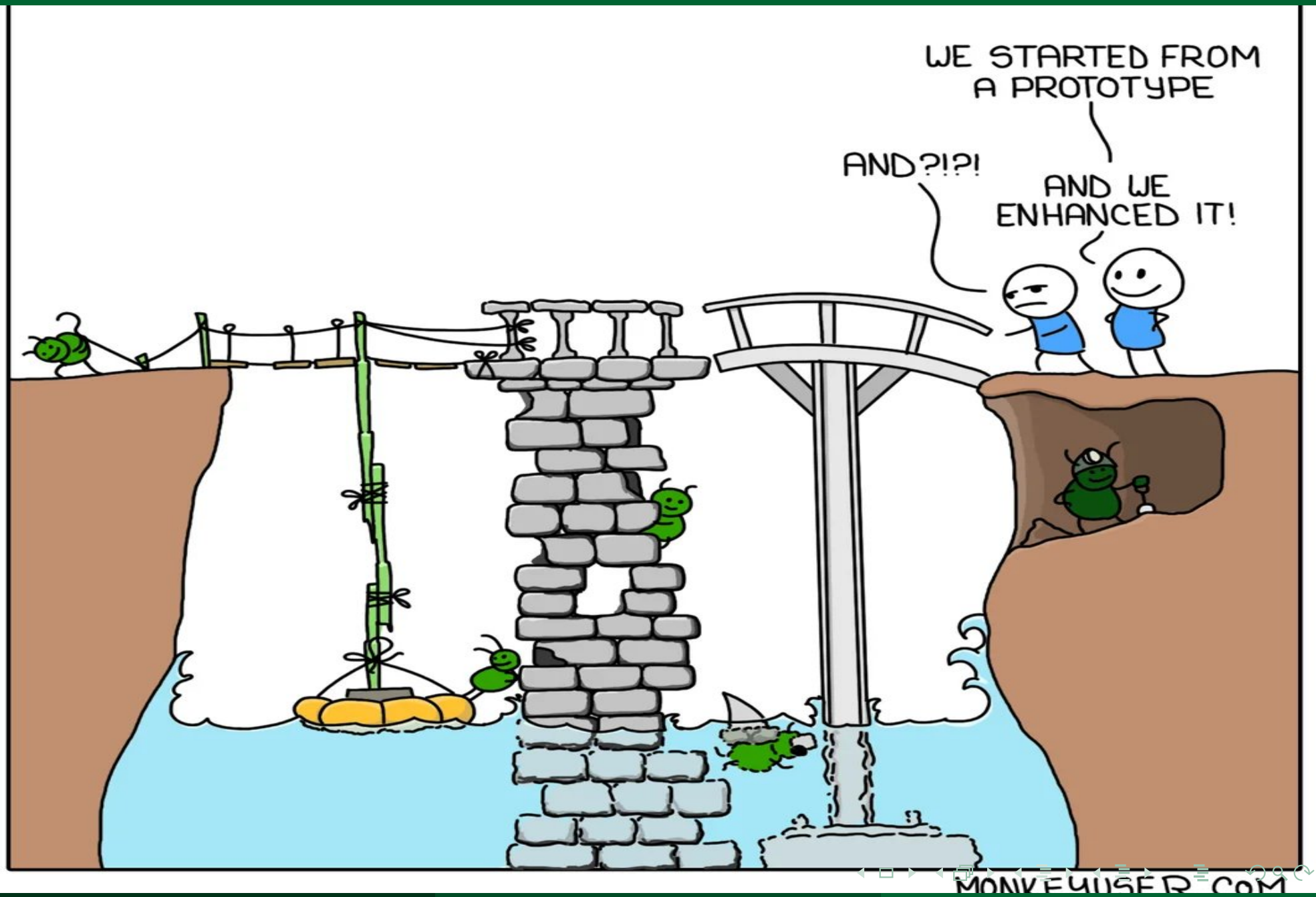
# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning
- Write your entire project in one script
- Have zero Separation of Concerns
- Make your code complex
- Lack of version control
- Lack of dependency tracking

# Practices to Avoid:

- Start writing code before you create a design or schematic for the program
- Avoid pure functions
- Fill your code with magic numbers and strings, hard-code in everything!
- Write spaghetti code
- Make everything sensitive to initial conditions
- Name your variables terms that have no meaning
- Write your entire project in one script
- Have zero Separation of Concerns
- Make your code complex
- Lack of version control
- Lack of dependency tracking
- Build a working prototype and never refactor it

# A Word on Prototyping:



# Example of Code out in the Wild 1:

```
me=fread(paste0(GENOME, '.filtered_disjoined_TEs.ecology.cytpatterns.out'))
me$TEID=substr(me$"9_usercol", 4,24)

colnames(me)[1:18]=c('chr', 'a', 'b', 'start', 'end', 'c', 'd', 'e', 'id', 'p_at', 'p_gc', 'A', 'C', 'G', 'T', 'N', 'other', 'seqlen')
colnames(me)[19:40]=c('CG', 'CAG', 'CTG', 'CCG', 'CAA', 'CAT', 'CAC', 'CTA', 'CTT', 'CTC', 'CCC', 'CCA', 'CCT', 'TTG', 'ATG', 'GTG', 'TAG', 'AAG', 'GAG', 'GGG', 'TGG', 'AGG')
colnames(me)[41:42]=c('TG', 'CA')

## dplyr change!
#mm=me %>% group_by(TEID) %>% summarize_if(.predicate=function(x) is.integer(x), .funs=funs('sum'))
mm=data.frame(me) %>% group_by(TEID) %>% dplyr::summarize_if(is.numeric, mean)
mm$start=NULL
mm$end=NULL
mm$sup=substr(mm$TEID, 1,3)
mm$fam=substr(mm$TEID, 1,8)


mm_sup=mm %>% group_by(sup) %>% summarize_if(.predicate=function(x) is.integer(x), .funs=funs('sum'))
mm_fam=mm %>% group_by(fam) %>% summarize_if(.predicate=function(x) is.integer(x), .funs=funs('sum'))

### summarize in proportions - GC, CG, CHG, CHH
mm$percGC=(mm$C+mm$G)/(mm$seqlen-mm$N)
mm$nCG=mm$CG/(mm$seqlen-mm$N)
mm$nCHG=(mm$CAG + mm$CTG + mm$CCG)/(mm$seqlen-mm$N)  ## these are normalized, because, for example, a CHG could be double counted in CHG and CG if it were CCG
mm$nCHH=(mm$CAA + mm$CAT + mm$CAC + mm$CTA + mm$CTT + mm$CTC + mm$CCC + mm$CCA + mm$CCT + mm$TTG + mm$ATG + mm$GTG + mm$TAG + mm$AAG + mm$GAG + mm$GGG + mm$TGG + mm$AGG)/(mm$seqlen-mm$N)/2
mm$nTG=(mm$TG + mm$CA)/(mm$seqlen-mm$N)

mm_fam$percGC=(mm_fam$C+mm_fam$G)/(mm_fam$seqlen-mm_fam$N)
mm_fam$nCG=mm_fam$CG/(mm_fam$seqlen-mm_fam$N)
mm_fam$nCHG=(mm_fam$CAG + mm_fam$CTG + mm_fam$CCG)/(mm_fam$seqlen-mm_fam$N)  ## these are normalized, because, for example, a CHG could be double counted in CHG and CG if it were CCG
mm_fam$nCHH=(mm_fam$CAA + mm_fam$CAT + mm_fam$CAC + mm_fam$CTA + mm_fam$CTT + mm_fam$CTC + mm_fam$CCC + mm_fam$CCA + mm_fam$CCT + mm_fam$TTG + mm_fam$ATG + mm_fam$GTG + mm_fam$TAG + mm_fam$AAG + mm
mm_fam$sup=substr(mm_fam$fam,1,3)

mm_sup$percGC=(mm_sup$C+mm_sup$G)/(mm_sup$seqlen-mm_sup$N)
mm_sup$nCG=mm_sup$CG/(mm_sup$seqlen-mm_sup$N)
mm_sup$nCHG=(mm_sup$CAG + mm_sup$CTG + mm_sup$CCG)/(mm_sup$seqlen-mm_sup$N)  ## these are normalized, because, for example, a CHG could be double counted in CHG and CG if it were CCG
mm_sup$nCHH=(mm_sup$CAA + mm_sup$CAT + mm_sup$CAC + mm_sup$CTA + mm_sup$CTT + mm_sup$CTC + mm_sup$CCC + mm_sup$CCA + mm_sup$CCT + mm_sup$TTG + mm_sup$ATG + mm_sup$GTG + mm_sup$TAG + mm_sup$AAG + mm

```

# Example of Code out in the Wild 2:

```
rm.ice=generateICE('segsites.bp', subset_rf, grid=data.frame(segsites.bp=seq(0,quantile(ind$segsites.bp, 0.95, na.rm=T), length.out=50)))
write.table(rm.ice, paste0('segsites.bp.', Sys.Date(), '.txt'), quote=F, sep='\t', row.names=F, col.names=T)
ggplot(rm.ice, aes(segsites.bp, yhat/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_c
ggplot(rm.ice, aes(segsites.bp, yhat.centered/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+
ggplot(rm.ice, aes(segsites.bp, yhat/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_c
ggplot(rm.ice, aes(segsites.bp, yhat.centered/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+

rm.ice=generateICE('anther_avg_chh', subset_rf, grid=data.frame(anther_avg_chh=seq(0,quantile(ind$anther_avg_chh, 0.95, na.rm=T), length.out=50)))
write.table(rm.ice, paste0('anther_avg_chh.', Sys.Date(), '.txt'), quote=F, sep='\t', row.names=F, col.names=T)
ggplot(rm.ice, aes(anther_avg_chh, yhat/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+
ggplot(rm.ice, aes(anther_avg_chh, yhat.centered/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color
ggplot(rm.ice, aes(anther_avg_chh, yhat/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+
ggplot(rm.ice, aes(anther_avg_chh, yhat.centered/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color

rm.ice=generateICE('tebp', subset_rf, grid=data.frame(tebp=seq(0,quantile(ind$tebp, 0.95, na.rm=T), length.out=50)))
write.table(rm.ice, paste0('tebp.', Sys.Date(), '.txt'), quote=F, sep='\t', row.names=F, col.names=T)
ggplot(rm.ice, aes(tebp, yhat/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_colo
ggplot(rm.ice, aes(tebp, yhat.centered/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  s
ggplot(rm.ice, aes(tebp, yhat/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_colo
ggplot(rm.ice, aes(tebp, yhat.centered/2/3.3e-8/1e6, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  s


rm.ice=generateICE('closest', subset_rf, grid=data.frame(closest=seq(0,quantile(ind$closest, 0.95, na.rm=T), length.out=50)))
write.table(rm.ice, paste0('closest.', Sys.Date(), '.txt'), quote=F, sep='\t', row.names=F, col.names=T)
ggplot(rm.ice, aes(closest, yhat/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_color
ggplot(rm.ice, aes(closest, yhat.centered/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  sc
ggplot(rm.ice, aes(closest, yhat/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_color
ggplot(rm.ice, aes(closest, yhat.centered/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  sc

rm.ice=generateICE('famsize', subset_rf)
write.table(rm.ice, paste0('famsize.', Sys.Date(), '.txt'), quote=F, sep='\t', row.names=F, col.names=T)
ggplot(rm.ice, aes(famsize, yhat/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_color
ggplot(rm.ice, aes(famsize, yhat.centered/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  sc
ggplot(rm.ice, aes(famsize, yhat/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  scale_color
ggplot(rm.ice, aes(famsize, yhat.centered/2/3.3e-8, color=sup)) + geom_line(aes(group = yhat.id), alpha = 0.2) + stat_summary(fun.y = mean, geom = "line",  size = 2, aes(group=sup, color=sup))+  sc

rm.ice=generateICE('percGC', subset_rf, grid=data.frame(percGC=seq(quantile(ind$percGC, 0.05, na.rm=T), quantile(ind$percGC, 0.95, na.rm=T), length.out=50)))
```

# Example of Code out in the Wild 3:

```
#subimp$category[grepl("flank_cg", subimp$feat)]='flank_cg_methylation'
#subimp$category[grepl("flank_chg", subimp$feat)]='flank_chg_methylation'
#subimp$category[grepl("flank_chh", subimp$feat)]='flank_chh_methylation'
#subimp$category[grepl("avg_cg", subimp$feat)]='te_cg_methylation'
#subimp$category[grepl("avg_chg", subimp$feat)]='te_chg_methylation'
#subimp$category[grepl("avg_chh", subimp$feat)]='te_chh_methylation'
#subimpsort=subimp %>% group_by(category) %>% summarize(sum=sum(meanimp)) %>% arrange(desc(sum))
#print(head(subimpsort))
#}

## colors for poster: paired for flank vs TE
#Create a custom color scale
myColors <- brewer.pal(12,"Paired")
myColors=myColors[c(1:6,8,10,11,12)]
names(myColors) <- c('flank_base_composition', 'TE_base_composition', 'flank_methylation_mnase', 'TE_methylation_mnase',
                     'flank_closest_gene_expression', 'TE_expression', 'TE_genes', 'TE_taxonomy', 'flank_selection', 'TE_features')
colScale <- scale_colour_manual(name = "feature",values = myColors)

## assign to categories
categories=data.frame(feature=imp$feat)
categories$category=NA
categories$category[categories$feature %in% c('fam', 'sup')]='TE_taxonomy'
categories$category[categories$feature %in% c('famsize', 'pieces', 'chr', 'helLCV3p', 'helLCV5p', 'TIRlen', 'avg_ltr_length','te_bp', 'tebp', 'tespan' )]='TE_features'
categories$category[grepl('_avg_c', categories$feature) | categories$feature %in% c('shoot_prop', 'shoot_bp','root_bp', 'root_prop', 'n_shoot_hs', 'n_root_hs') ]='TE_methylation_mnase'
categories$category[categories$feature %in% c('nCG', 'nCHG', 'nCHH', 'nTG', 'percGC') ]='TE_base_composition'
categories$category[categories$feature %in% c('GAG', 'AP', 'RT', 'RNaseH', 'INT', 'ENV', 'CHR', 'auton', 'pol', 'orfAA', 'helprot', 'tirprot', 'rveprot', 'tpaseprot', 'GAGfam', 'APfam', 'INTfam', '
categories$category[grepl('_[[:digit:]]+', categories$feature) | grepl('h3k9me2_[[:digit:]]+', categories$feature) | categories$feature %in% c('flank_h3k9me2', 'flank_cg', 'flank_chg', 'flank_chh')
categories$category[grepl('^avg_c', categories$feature)]='flank_base_composition'
categories$category[grepl('cm', categories$feature) | grepl('segsites', categories$feature) | categories$feature %in% c('closest', 'closestgenesyntenic', 'closest.syntenic', 'ingene', 'subgenome',
categories$category[grepl('^gene_', categories$feature) | grepl('^syntenicgene_', categories$feature) ]='flank_closest_gene_expression'
categories$category[grepl('^TEfam', categories$feature) | grepl('unique', categories$feature)]='TE_expression'

## new ways to get at this - quick fix :(
categories$category[categories$feature %in% c('percGC_1kbflank','nCG_1kbflank','nCHG_1kbflank','nCHH_1kbflank','flank_bp', 'nTG_1kb_flank')]='flank_base_composition'
categories$category[categories$feature %in% c('flank_n_root_hs','flank_root_bp','flank_root_prop','flank_n_shoot_hs','flank_shoot_bp','flank_shoot_prop')]='flank_methylation_mnase'
categories$category[categories$feature %in% c('segsites.bp', 'disruptor', 'disruptor.samefam', 'disruptor.samesup')]='TE_features'
categories[is.na(categories$category),]
```

# Code & Data:

- Write code that is easy to read and understand
  - Give descriptive, meaningful names to your variables and functions
  - Comment your code at the top: purpose, expected usage, example inputs/outputs
  - Record dependencies, imports, and constants at the top
  - Write a docstring for every function
- Do not tamper with original data files
  - Do not make changes by hand; automate everything, document everything

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`
    - all of your code and scripts

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`
    - all of your code and scripts
  - `doc`

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`
    - all of your code and scripts
  - `doc`
    - research & literature notes, analysis notes, any reports

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`
    - all of your code and scripts
  - `doc`
    - research & literature notes, analysis notes, any reports
  - `results`

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`
    - all of your code and scripts
  - `doc`
    - research & literature notes, analysis notes, any reports
  - `results`
    - Figures, graphs, and any filtered data or output data.

Adapted from Arjun Krishnan's computational biology course

# Organizing Your Research:

- **Why is this important?**
  - Partitioning your data, code, results and other items into a directory structure helps you stay organized.
  - You don't want to accidentally delete your data, or have it overwritten by a new file.
  - Someone else should be able to look at your directory structure and understand what you are doing.

- **Sample Project Directory:**
  - `data`
    - primary data
  - `src`
    - all of your code and scripts
  - `doc`
    - research & literature notes, analysis notes, any reports
  - `results`
    - Figures, graphs, and any filtered data or output data.
  - `Makefile` or workflow script that catalogs your commands and makes re-running your analyses easy.

Adapted from Arjun Krishnan's computational biology course

# What is Git & Why Should I Use It?

- **What is Git?**

# What is Git & Why Should I Use It?

- **What is Git?**
  - Git is a distributed version control system

# What is Git & Why Should I Use It?

- **What is Git?**
  - Git is a distributed version control system
- **What can Git do for me?**

# What is Git & Why Should I Use It?

- **What is Git?**
  - Git is a distributed version control system
- **What can Git do for me?**
  - Git is a tool that helps you to track changes in your code

# What is Git & Why Should I Use It?

- **What is Git?**
  - Git is a distributed version control system
- **What can Git do for me?**
  - Git is a tool that helps you to track changes in your code
  - Git is a tool that helps with collaboration

# What is Git & Why Should I Use It?

- **What is Git?**
  - Git is a distributed version control system
- **What can Git do for me?**
  - Git is a tool that helps you to track changes in your code
  - Git is a tool that helps with collaboration
  - Git is a time machine

# What is Git & Why Should I Use It?

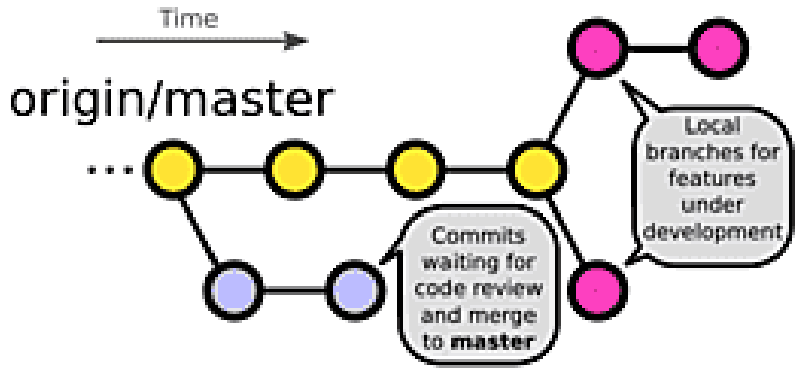
- **What is Git?**
  - Git is a distributed version control system
- **What can Git do for me?**
  - Git is a tool that helps you to track changes in your code
  - Git is a tool that helps with collaboration
  - Git is a time machine
- **Where can I learn more?**

# What is Git & Why Should I Use It?

- **What is Git?**
  - Git is a distributed version control system
- **What can Git do for me?**
  - Git is a tool that helps you to track changes in your code
  - Git is a tool that helps with collaboration
  - Git is a time machine
- **Where can I learn more?**
  - `https://git-scm.com/book/en/v2`
  - `https://learngitbranching.js.org/`
  - `https://www.atlassian.com/git/tutorials`
  - `https://docs.github.com/en/get-started/getting-started-with-git`
  - Youtube!

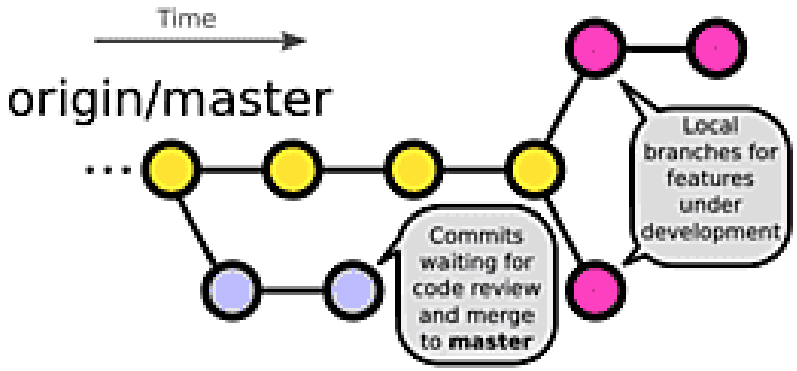
# A Note on Git Branches and Development:

- The `master` branch is sacred. Try to keep it clean and in a working state.



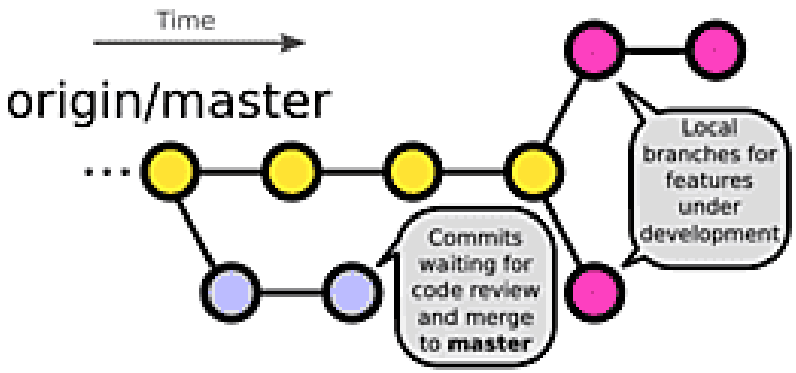
# A Note on Git Branches and Development:

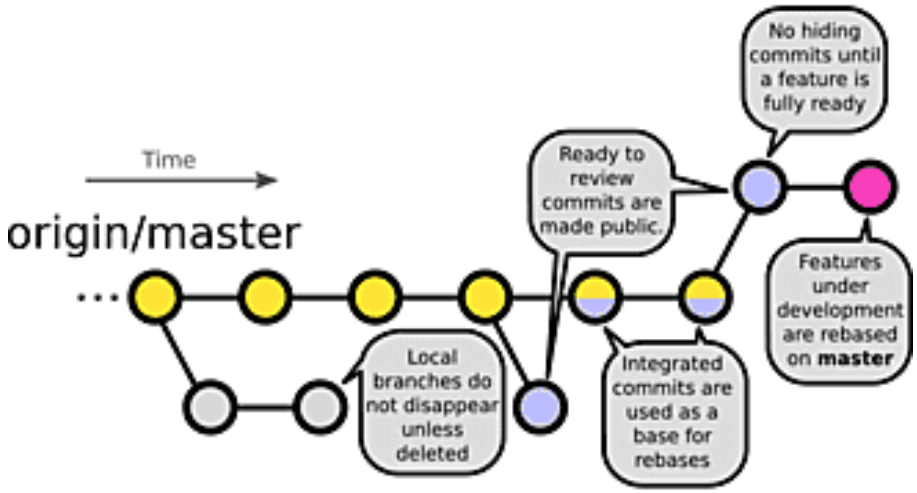
- The `master` branch is sacred. Try to keep it clean and in a working state.
- Use branches to develop new features or fix bugs. Examples:
  - `f/new_barplot`
  - `b/bugfix_calculation`
  - `f/load_data`



# A Note on Git Branches and Development:

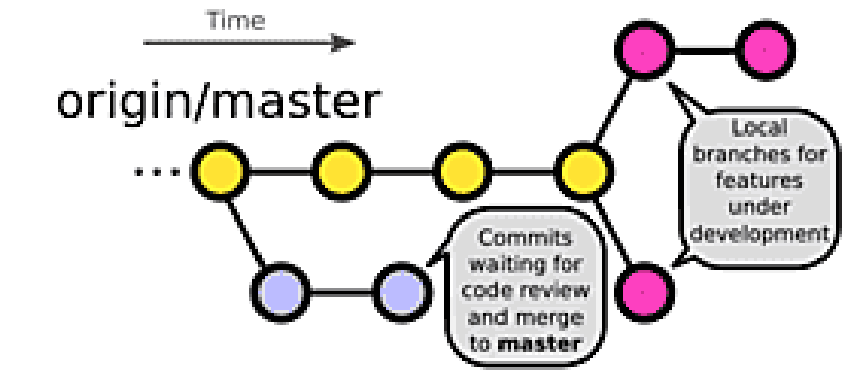
- The `master` branch is sacred. Try to keep it clean and in a working state.
- Use branches to develop new features or fix bugs. Examples:
  - `f/new_barplot`
  - `b/bugfix_calculation`
  - `f/load_data`
- Later, integrate your changes into the `master` branch with a pull request.
  - Before merging, make sure your code is well-documented.
  - You should think about performing a `git rebase` to clean up your commit history. It doesn't help anybody if your commit history is a mess, and filled with a million barely consequential commits.

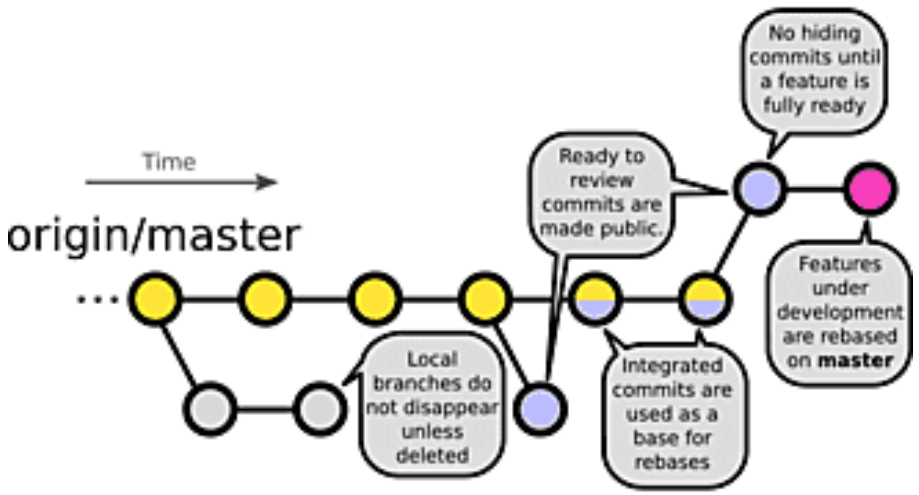


# A Note on Git Branches and Development:

- The `master` branch is sacred. Try to keep it clean and in a working state.
- Use branches to develop new features or fix bugs. Examples:
  - `f/new_barplot`
  - `b/bugfix_calculation`
  - `f/load_data`
- Later, integrate your changes into the `master` branch with a pull request.
  - Before merging, make sure your code is well-documented.
  - You should think about performing a `git rebase` to clean up your commit history. It doesn't help anybody if your commit history is a mess, and filled with a million barely consequential commits.
- More reading at: Git Cactus Model



# Package Management in Python:

- **What is package management?**

# Package Management in Python:

- **What is package management?**
  - Package management is the process of installing, upgrading, configuring, and removing software packages.

# Package Management in Python:

- **What is package management?**
  - Package management is the process of installing, upgrading, configuring, and removing software packages.
- **Why is package management important?**

# Package Management in Python:

- **What is package management?**
  - Package management is the process of installing, upgrading, configuring, and removing software packages.
- **Why is package management important?**
  - Package management is important because it helps you to manage dependencies, and it helps minimize confusion over what version of a package you are using.

# Package Management in Python:

- **What is package management?**
  - Package management is the process of installing, upgrading, configuring, and removing software packages.
- **Why is package management important?**
  - Package management is important because it helps you to manage dependencies, and it helps minimize confusion over what version of a package you are using.
- **How do I manage packages in Python?**

# Package Management in Python:

- **What is package management?**
  - Package management is the process of installing, upgrading, configuring, and removing software packages.
- **Why is package management important?**
  - Package management is important because it helps you to manage dependencies, and it helps minimize confusion over what version of a package you are using.
- **How do I manage packages in Python?**
  - You can use `pip` to install packages, and you can use `requirements.txt` to keep track of your dependencies.

# Package Management in Python:

- **What is package management?**
  - Package management is the process of installing, upgrading, configuring, and removing software packages.
- **Why is package management important?**
  - Package management is important because it helps you to manage dependencies, and it helps minimize confusion over what version of a package you are using.
- **How do I manage packages in Python?**
  - You can use `pip` to install packages, and you can use `requirements.txt` to keep track of your dependencies.
  - Please check out `https://docs.python.org/3/library/venv.html` for information on how to create a virtual environment.

# Python & R Best Practices:

- Resist the urge to hard-code in everything!

# Python & R Best Practices:

- Resist the urge to hard-code in everything!
- If you need to do a lot of table formatting before your analysis, separate that from your analysis code.

# Python & R Best Practices:

- Resist the urge to hard-code in everything!
- If you need to do a lot of table formatting before your analysis, separate that from your analysis code.
- Use functions to delineate stages of your analysis

# Python & R Best Practices:

- Resist the urge to hard-code in everything!
- If you need to do a lot of table formatting before your analysis, separate that from your analysis code.
- Use functions to delineate stages of your analysis
- Live demo of a Python script that is a decent start

# Python & R Best Practices:

- Resist the urge to hard-code in everything!
- If you need to do a lot of table formatting before your analysis, separate that from your analysis code.
- Use functions to delineate stages of your analysis
- Live demo of a Python script that is a decent start
- Follow a style guide!

# Python & R Best Practices:

- Resist the urge to hard-code in everything!
- If you need to do a lot of table formatting before your analysis, separate that from your analysis code.
- Use functions to delineate stages of your analysis
- Live demo of a Python script that is a decent start
- Follow a style guide!
  - Python: `https://github.com/psf/black`

# Python & R Best Practices:

- Resist the urge to hard-code in everything!
- If you need to do a lot of table formatting before your analysis, separate that from your analysis code.
- Use functions to delineate stages of your analysis
- Live demo of a Python script that is a decent start
- Follow a style guide!
  - Python: `https://github.com/psf/black`
  - R: `https://style.tidyverse.org/`

# IDEs & Editors:

- **VSCode:** Probably the best choice for most people, as it has a lot of features, including a terminal, and a built-in Git interface. You can also work on a collaborative coding session with others remotely, which is good for pair-programming.

- **Vim:** My personal choice, but it has a steep learning curve. It is very powerful, and is great for working on remote servers. It has a lot of customizability and plugins.

- **Emacs:** Another powerful editor, and it also has a steep learning curve. Competitor to Vim, also very customizable

# Makefiles, Snakemake, Nextflow:

- These are tools that help you to automate your workflow. Essentially, you define a series of rules that dictate how your data is processed, and you can define dependencies between rules.
- If you have a lot of steps in your analysis, these tools can help you to organize your workflow, and make it easier to reproduce your results.
- Nextflow & Snakemake are popular in the bioinformatics community, but Makefiles are the original technology, and more general/appropriate if you become a software developer. I have heard a lot of good things about Nextflow.
- Their implementation is beyond the scope of this presentation, but I wanted to mention them as they are very useful tools for reproducibility.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.
  - `https://adv-r.hadley.nz/debugging.html` is a good resource for approaches.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.
  - `https://adv-r.hadley.nz/debugging.html` is a good resource for approaches.
- Googling is your friend! You will find that most of the time, someone else has had the same problem as you. Try to word your problem generally.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.
  - `https://adv-r.hadley.nz/debugging.html` is a good resource for approaches.
- Googling is your friend! You will find that most of the time, someone else has had the same problem as you. Try to word your problem generally.
- Take the time to learn what each error actually means, and how to interpret them. You will surely run into them again. Don't focus so much on trying to make the problem go away, but rather understand why it is happening.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.
  - `https://adv-r.hadley.nz/debugging.html` is a good resource for approaches.
- Googling is your friend! You will find that most of the time, someone else has had the same problem as you. Try to word your problem generally.
- Take the time to learn what each error actually means, and how to interpret them. You will surely run into them again. Don't focus so much on trying to make the problem go away, but rather understand why it is happening.
- If you are using a function, look up the documentation to make sure you have the parameters correct.

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.
  - `https://adv-r.hadley.nz/debugging.html` is a good resource for approaches.
- Googling is your friend! You will find that most of the time, someone else has had the same problem as you. Try to word your problem generally.
- Take the time to learn what each error actually means, and how to interpret them. You will surely run into them again. Don't focus so much on trying to make the problem go away, but rather understand why it is happening.
- If you are using a function, look up the documentation to make sure you have the parameters correct.
- Try to break your problem down into smaller parts, and test each part individually, this is where functions come in handy. No blob code!

# Debugging:

- Debugging is a fact of life and you will spend a lot of time doing it.
- Debugging via `print` statements is a good way to start, but be careful not to fall into bad habits. Print statements will not get you very far with complex problems.
- More advanced debugging should be done with a debugger such as `pdb` in Python, R's built-in debugger, or an IDE's debugger.
  - `https://adv-r.hadley.nz/debugging.html` is a good resource for approaches.
- Googling is your friend! You will find that most of the time, someone else has had the same problem as you. Try to word your problem generally.
- Take the time to learn what each error actually means, and how to interpret them. You will surely run into them again. Don't focus so much on trying to make the problem go away, but rather understand why it is happening.
- If you are using a function, look up the documentation to make sure you have the parameters correct.
- Try to break your problem down into smaller parts, and test each part individually, this is where functions come in handy. No blob code!
- If you are working on a large dataset, and need to load it in before you can start working, try to work on a subset of the data first so that you can iterate faster between errors.

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.
- **Provide the steps you have taken to try to solve the problem:** This is important, as it shows that you have put in effort to solve the problem, and you aren't just asking someone to do your work for you.

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.
- **Provide the steps you have taken to try to solve the problem:** This is important, as it shows that you have put in effort to solve the problem, and you aren't just asking someone to do your work for you.
- **Create a structured SOS:** This makes it easier for people to help you. I like to divide into sections:

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.
- **Provide the steps you have taken to try to solve the problem:** This is important, as it shows that you have put in effort to solve the problem, and you aren't just asking someone to do your work for you.
- **Create a structured SOS:** This makes it easier for people to help you. I like to divide into sections:
  - Problem:

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.
- **Provide the steps you have taken to try to solve the problem:** This is important, as it shows that you have put in effort to solve the problem, and you aren't just asking someone to do your work for you.
- **Create a structured SOS:** This makes it easier for people to help you. I like to divide into sections:
  - Problem:
  - Context:

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.
- **Provide the steps you have taken to try to solve the problem:** This is important, as it shows that you have put in effort to solve the problem, and you aren't just asking someone to do your work for you.
- **Create a structured SOS:** This makes it easier for people to help you. I like to divide into sections:
  - Problem:
  - Context:
  - Attempted Solutions:

# Asking for Help:

- **Be specific:** The more specific you are, the more likely you are to get a helpful response.
- **Provide a minimal reproducible example:** This is a small piece of code that demonstrates your problem. This is very helpful for people trying to help you, as they often don't have the context of your entire project.
- **Provide the version of the software you are using:** Different versions of software can have different behavior.
- **Provide the steps you have taken to try to solve the problem:** This is important, as it shows that you have put in effort to solve the problem, and you aren't just asking someone to do your work for you.
- **Create a structured SOS:** This makes it easier for people to help you. I like to divide into sections:
  - Problem:
  - Context:
  - Attempted Solutions:
  - Desired Outcome:

# Teaching Yourself:

- **Read the manual and documentation:** This is the most important thing you can do. Don't just read the part that you think is immediately relevant to your problem. For most command-line tools you should be able to run `man <command>` to get the manual.
- **Familiarize yourself with the standard library of a program:** I can't tell you how many times I have seen people write code that is already implemented in the standard library of a language (`argparse`, `os` in Python)
  - For non-standard library but common packages relevant to your field, master the package, as a lot of the functions are probably already implemented. E.g Python Pandas, R dplyr, for data manipulation.

# References & Further Reading:

- *AntiPatterns — Refactoring Software, Architectures, and Projects in Crisis*
  - This book is like $15, and while dated, it is still very relevant, as it has a lot of good advice on how to tackle and communicate problems. I'm glad I purchased it.
    A lot of the material I discussed today is from this book.
- Boot.dev (website)
  - Costs money, but has a lot of very good resources for Python, Docker, Git, and more.
  - The lessons are "gamified" and are very interactive.
- *The Last Algorithms Course You'll Need*
  - Free at `https://frontendmasters.com/courses/algorithms/`
  - Has a lot of good information on what common data structures and algorithms are. These things give you much greater perspective as a programmer. Not a lot of attention in the data science commmunity is paid to performance (such as Big O notation), and people fundamentally don't understand basic data structures and their implications, i.e why you shouldn't append a million times to a list.
- *The Craft of Scientific Writing* by Michael Alley
- Arjun Krishnan's teaching materials
  `https://github.com/krishnanlab/teaching/tree/master`
  - This is extremely high quality material.
  - There are a lot of biology-related examples and how-to guides.