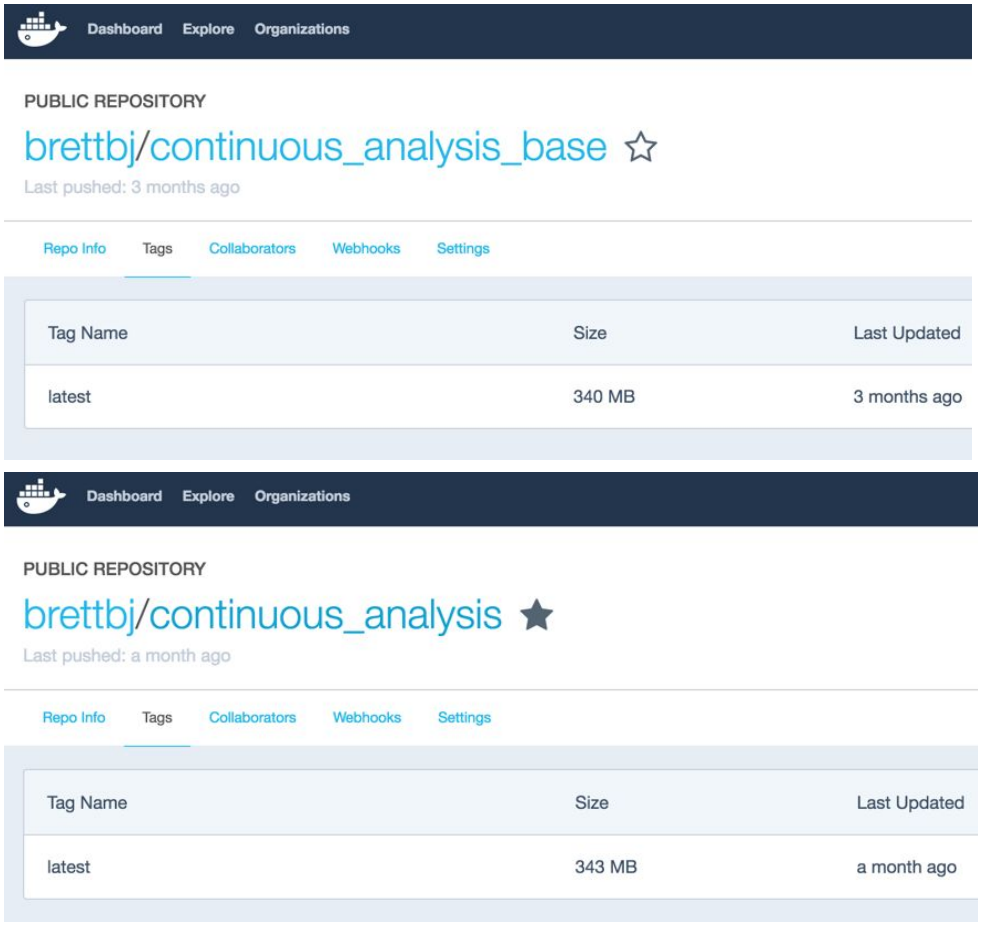
Reproducible Computational Workflows with Continuous Analysis - Supplemental Materials

Contents:

1 - Continuous Analysis for Readers and Reviewers

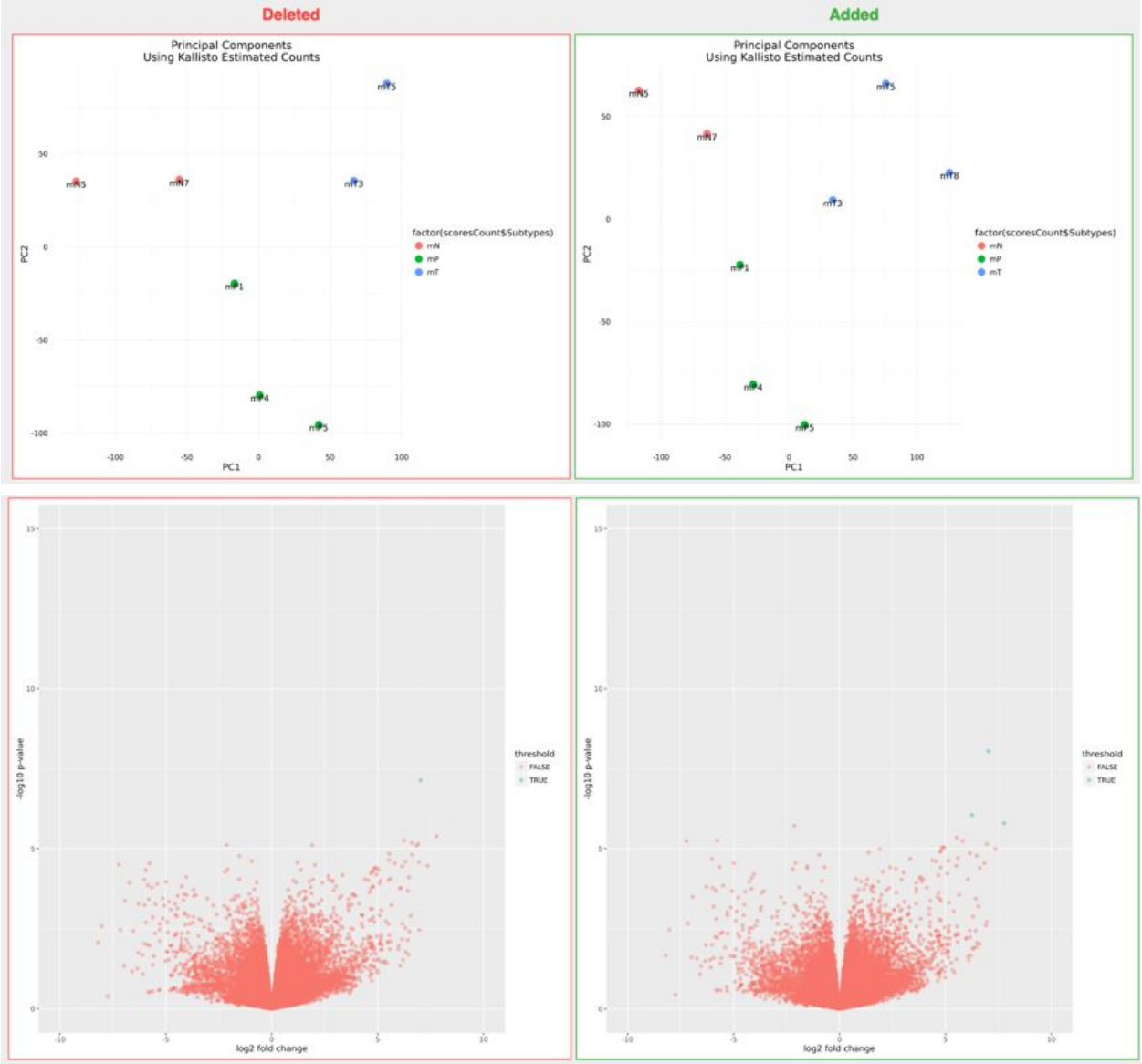
Continuous analysis produces three main outputs useful to readers and reviewers.

1. Docker containers of the exact environment the analysis was run in. These allow readers and reviewers to either rerun analysis or perform additional analysis from the published base. A helpful guide for getting started working with docker is available on Docker's website - https://docs.docker.com/ (choose your preferred operating system.)



Supplemental Figure 1. Docker images available for the continuous analysis setup example before (https://hub.docker.com/r/brettbj/continuous_analysis_base/) and after (https://hub.docker.com/r/brettbj/continuous_analysis/).

2. Regenerated figures for each continuous analysis run. These figures are synchronized to the exact code used to generate them. Popular repository software systems, such as Github or Bitbucket, make it easy to view changes in order to see the effect source code changes have on final figures.



Supplemental Figure 2. A.) Adding a sample to a sample RNA-seq differential expression analysis maintains a similar looking PCA plot. **B.)** Adding a sample adds several significant differentially expressed genes. (live figure version: https://git.io/v6LDp)

3. Continuous integration logs that show the exact analysis performed. These logs display intermediate steps and can be used as an audit of what was done without the need to re-execute source code. A complete example is available at: https://raw.githubusercontent.com/greenelab/continuous_analysis_phylo/master/references/full_logs.txt

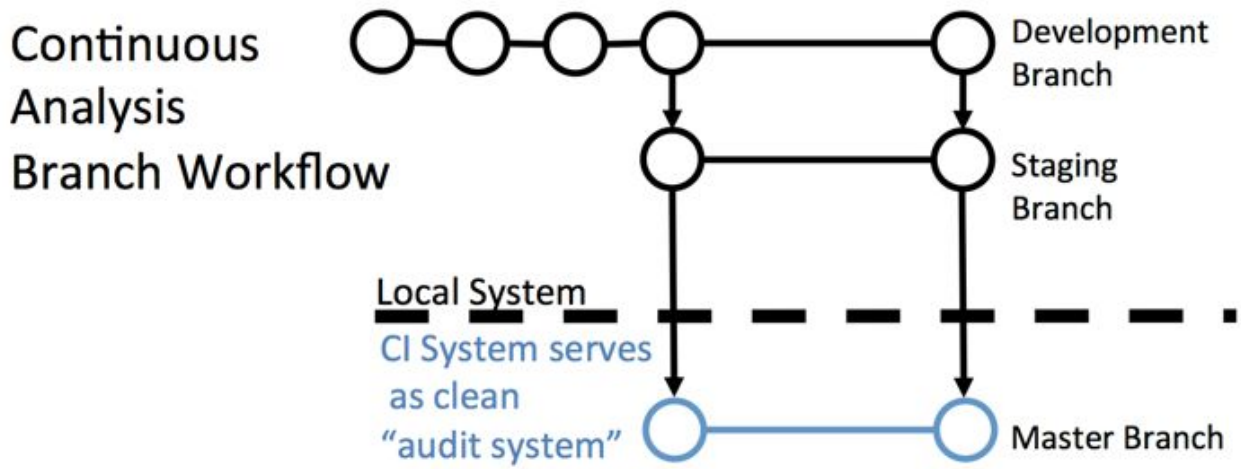
```
$ kallisto quant -i mm10  -o kallisto_output/SRR1654628 --bootstrap-samples=10 SRR1654628_1.fastq.gz
SRR1654628_2.fastq.gz -t 28

[quant] fragment length distribution will be estimated from the data
[index] k-mer length: 31
[index] number of targets: 35,026
[index] number of k-mers: 64,110,878
[index] number of equivalence classes: 81,212
[quant] running in paired-end mode
[quant] will process pair 1: SRR1654628_1.fastq.gz
                             SRR1654628_2.fastq.gz
[quant] finding pseudoalignments for the reads ... done
[quant] processed 54,088,667 reads, 35,922,079 reads pseudoaligned
[quant] estimated average fragment length: 175.781
[   em] quantifying the abundances ... done
[   em] the Expectation-Maximization algorithm ran for 1,351 rounds
[~warn] number of threads (28) greater than number of bootstraps
[~warn] (cont'd) updating threads to number of bootstraps 10
[bstrp] number of EM bootstraps complete: 1
[bstrp] number of EM bootstraps complete: 2
[bstrp] number of EM bootstraps complete: 3
[bstrp] number of EM bootstraps complete: 4
[bstrp] number of EM bootstraps complete: 5
[bstrp] number of EM bootstraps complete: 6
[bstrp] number of EM bootstraps complete: 7
[bstrp] number of EM bootstraps complete: 8
[bstrp] number of EM bootstraps complete: 9
[bstrp] number of EM bootstraps complete: 10
```

Supplemental Figure 3. Part of a continuous integration log showing quantification of the abundances of RNA transcripts from RNA-seq data using Kallisto.

2A - Setting up Continuous Analysis - *High Level Decisions*

Branch Strategy

We recommend using a git branching strategy as a part of continuous analysis. In this branching strategy, new code development takes place on one branch (or many feature branches). When the new feature or code change is complete, merge this branch into staging. We set the continuous integration hook to watch the staging branch; when the staging branch is committed the continuous analysis process runs. When continuous analysis succeeds it pushes the code and updated analysis in the form and figures and results to master. The branches to run continuous integration for can be specified in the branches section of the .yml configuration file.



Supplemental Figure 4. Code changes are made on development branches. When completed, changes are merged into the staging branch and continuous integration runs. If the continuous integration process succeeds, changes are merged into the master branch and pushed along with regenerated figures and results.

Continuous Integration Service Options

Full service continuous integration services offer the fastest startup but most are limited by their ability to scale to large data or computations. In addition, these services are shared tenant providers and cannot be used with private data. Full services providers with free options currently include Shippable, Wercker, Travis CI, and Circle CI.

When using private data a private continuous integration server can be easily configured on a local machine using Drone. Examples in this work use Drone 0.4. When computational needs require more resources Drone can be used to install a private continuous integration server in the cloud or on a cluster compatible with docker images.

```
# choose the base docker image
image: brettbj/continuous_analysis_base
script:
    # run tests
    # perform analysis

# publish results
publish:
    docker:
        # docker details
```

Supplemental Figure 5. Example .yml file structure, choose your docker image, run tests, perform analysis and then publish results.

Continuous Integration Scripting

Each continuous integration configuration first downloads the source code repository and then runs a specified script (typically in the form of a .yml script). This script resembles a bash script and allows a user to download and organize their data and then execute their source code. This script is also used to merge the results into the master branch and push the changes back to the repository host.

2B - Setting up Continuous Analysis - *Detailed Walkthrough*

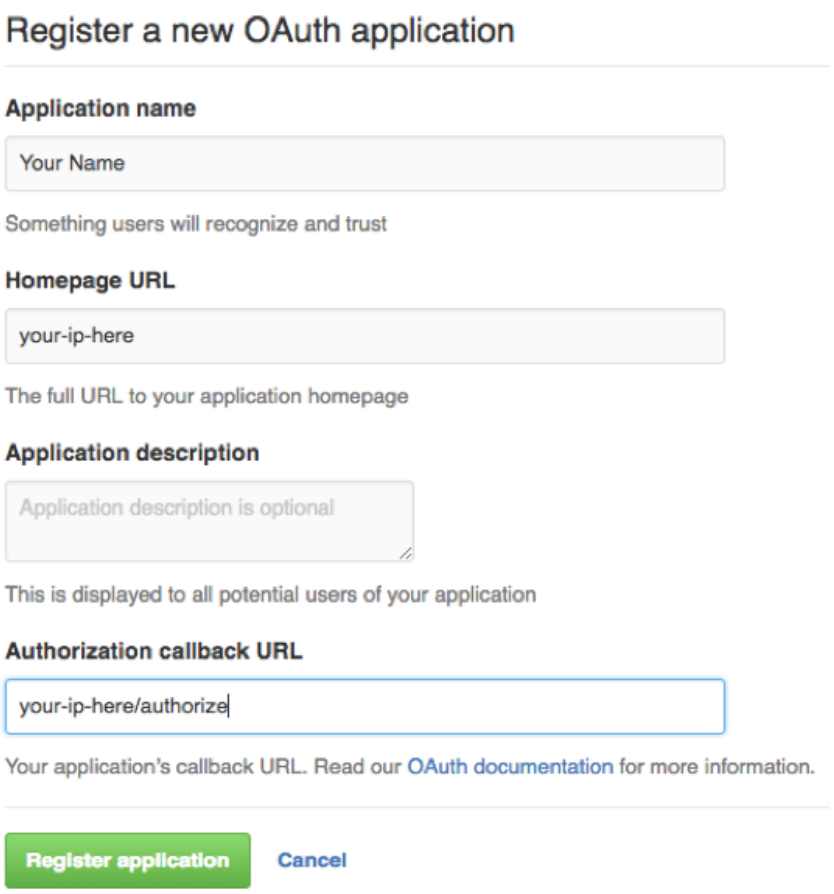
We show Continuous Analysis set up on a local or private cluster using the open source Drone continuous integration service. Up to date instructions are avaiable at http://readme.drone.io/setup/overview/.

Install Docker on host machine - Linux, Mac, Windows
Pull the drone image via docker

```
sudo docker pull drone/drone:0.4
```

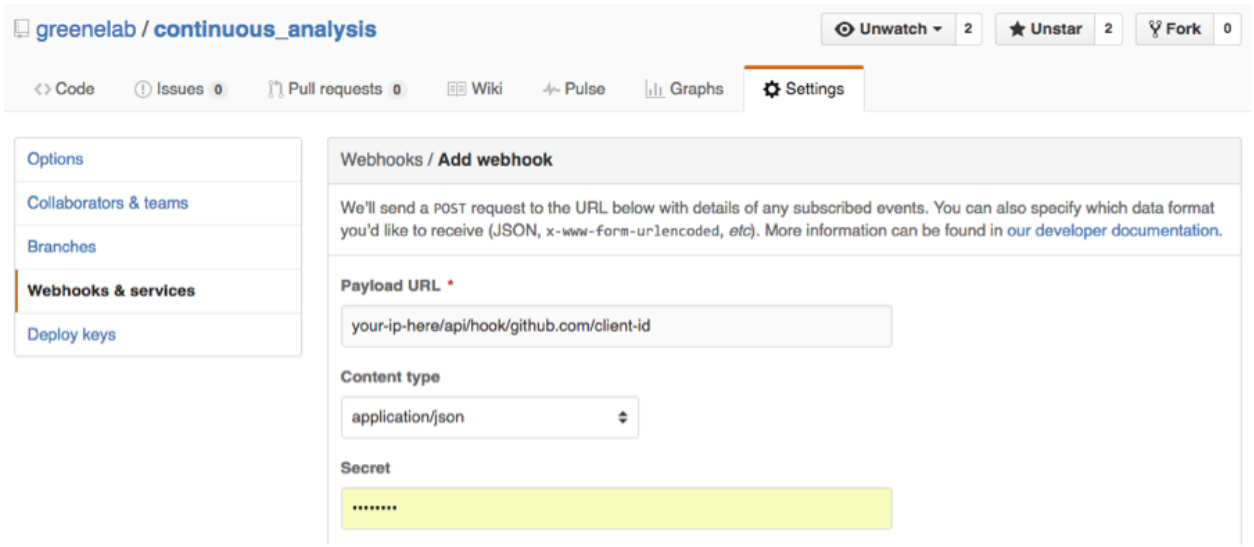
Create a new application at - https://github.com/settings/developers - with your hosts ip address in the homepage URL and the authorization callback followed by /authorize/

```
Homepage URL: http://YOUR-IP-HERE/
Callback URL: http://YOUR-IP-HERE/authorize/
```



Supplemental Figure 6. Register an application with github, using your IP address as the homepage url and your IP address followed by “/authorize” in the authorization callback URL.

Add a webhook in to notify the continuous integration server of any updates pushed to the repository.



Supplemental Figure 7. Add a webhook in your repository to initiate CI when pushing. The payload URL should be in the format of your-ip/api/hook/github.com/client-id.

Create a configuration file on your local machine at (/etc/drone/dronerc), filling in the client information.

```
REMOTE_DRIVER=github
REMOTE_CONFIG=https://github.com?client_id=....&client_secret….
```

Access the drone control panel at your-ip-address/login and click the github button to login with github.
After logging in, choose the repository you would like to use continuous analysis on. Add a .drone.yml file to your repository and continuous integration will run with each new push to the repository.

Example .drone.yml files are available at:

https://github.com/greenelab/continuous_analysis/blob/master/.drone.yml
https://github.com/greenelab/continuous_analysis_phylo/blob/master/.drone.yml
https://github.com/greenelab/continuous_analysis_rnaseq/blob/master/.drone.yml

Save static versions of the docker images before and after analysis. These can be uploaded to services like Zenodo and Figshare to receive DOIs.

```
docker save brettbj/continuous_analysis_base > continuous_analysis_base
docker save brettbj/continuous_analysis > continuous_analysis
```

https://dx.doi.org/10.6084/m9.figshare.3545156.v1

Once a continuous integration service is set up and connected to a github account, only the webhook, selection of the repository and configuration of the .drone.yml file are necessary for subsequent projects.

3 - Sample Project - Continuous Analysis for phylogeny tree building

Overview:
In this example we align 5 mRNA sequences and use these alignments to build phylogenies.

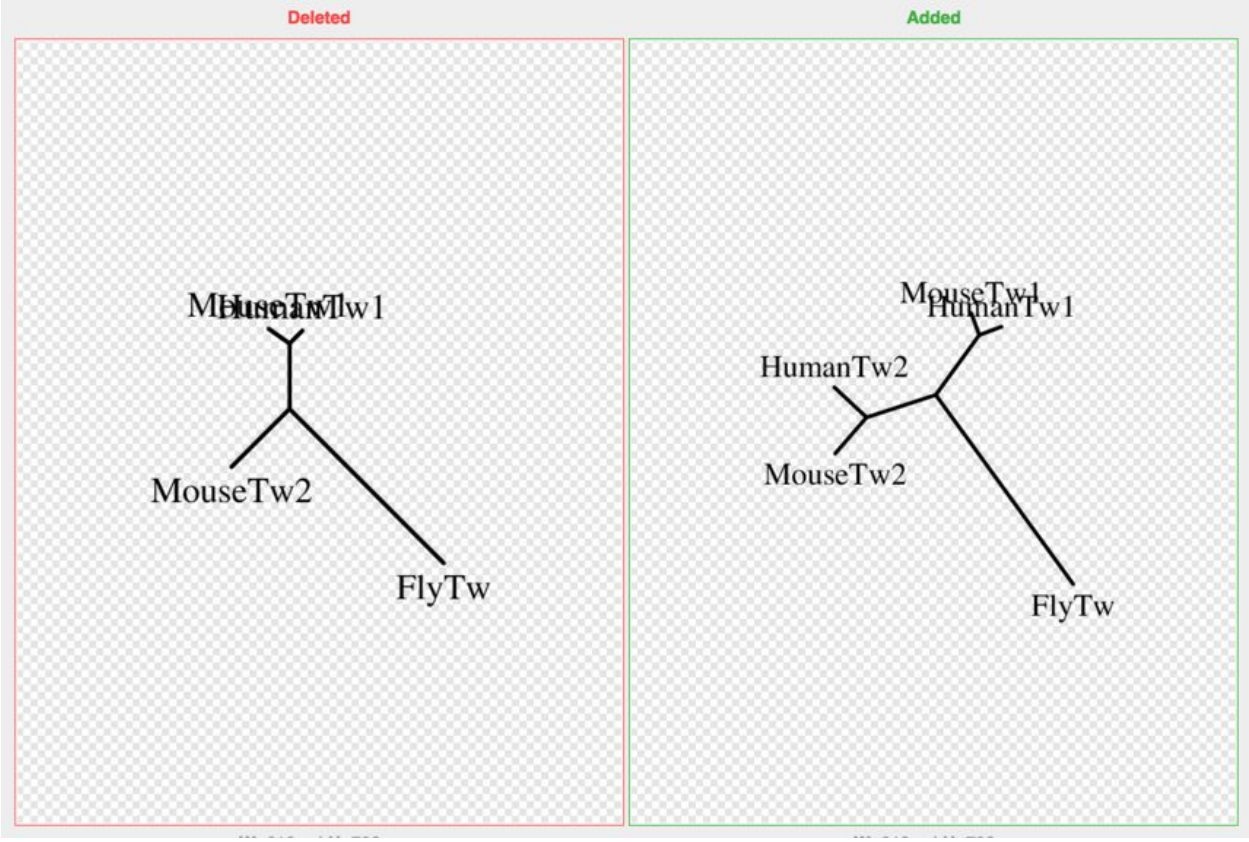
Link: https://github.com/greenelab/continuous_analysis_phylo

Analysis Setup:

1. We look at 5 mRNA sequences (findable at: http://www.ncbi.nlm.nih.gov/nuccore)
    - Twist - Fly (NM_079092, splice form A)
    - Twist1 - Human (NM_000474)
    - Twist1 - Mouse (NM_011658)
    - Twist2 - Human (NM_057179) - Added in second commit
    - Twist2 - Mouse (NM_007855)

    And load the sequences into twist.fasta
2. Align the sequences using MAFFT.
3. Convert to PHYLIP interleaved format using EMBOSS Seqret.
4. Calculate the maximum parsimony tree for the sequences using PHYLIP DNAPARS.
5. Draw a representation of this tree using PHYLIP drawtree.
6. Use Seqboot to assess the robustness of the generated tree.
7. Determine the consensus tree from the bootstrapped trees.

Sample Results:



Supplemental Figure 8. The effect of adding the HumanTw2 sequence to the constructed phylogenetic tree.

The docker image to rerun the analysis is available at: https://hub.docker.com/r/brettbj/continuous_analysis_phylo/

Static docker images are also available via Figshare https://dx.doi.org/10.6084/m9.figshare.3545156.v1

The addition of the HumanTw2 gene: https://git.io/v6kIk

The continuous analysis run after the addition: https://git.io/v6kIY

Continuous Analysis Configuration for this example:

This sample project uses the cloud service Digital Ocean as a host for continuous analysis. DigitalOcean provides an easy way to launch a cloud-based private continuous integration service. Instructions below were adapted from https://www.digitalocean.com/community/tutorials/how-to-use-the-drone-one-click-application-image.

1. Create a drone droplet, selecting Drone on Ubuntu 14.04 from the applications tab.
2. Create a new application at - https://github.com/settings/developers - with your hosts ip address in the homepage URL and the authorization callback followed by /api/auth/github.com

   ```
   Homepage URL: http://YOUR-IP-HERE/
   Callback URL: http://YOUR-IP-HERE/authorize/
   ```

3. Take note of the Client ID and Client Secret and log into your new droplet via ssh. You'll be asked a few questions, for simplest configuration choose automatic configuration. You'll be prompted to choose your code repository and enter your Client ID and Client secret.
4. You're ready to start by going to: http://YOUR-IP-HERE/login. After logging in you can select the repository you would like to use continuous analysis with.
5. The script run during continuous analysis can be viewed at: https://github.com/greenelab/continuous_analysis_phylo/blob/master/.drone.yml
6. Save static versions of the docker images before and after analysis. These can be uploaded to services like Zenodo and Figshare to receive DOIs.

   docker save brettbj/continuous_analysis_phylo > continuous_analysis_phylo
   docker save brettbj/continuous_analysis_phylo_post > https://dx.doi.org/10.6084/m9.figshare.3545156.v1

4 - Sample Project - Continuous Analysis for RNA-Seq differential expression analysis

Overview:
In this example we perform differential expression analysis of RNA-Seq data. This data is from Boi et al.[51], we follow and reuse source code published by Balli[52]. The original analysis and source code is available at - https://benchtobioinformatics.wordpress.com/2015/07/10/using-kallisto-for-gene-expression-analysis-of-published-rnaseq-data/

In this analysis the authors examine organoid models of pancreatic cancer in mice. To do this they generated organoids from three different tissues: normal pancreas (mN), early stage lesions (mP), or pancreatic adenocarcinoma (mT). The authors performed RNA-Seq on these organoids and published them to the Gene Expression Omnibus (http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE63348) and the Sequence Read Archive (http://www.ncbi.nlm.nih.gov/sra?term=SRP049959).
Link: https://github.com/greenelab/continuous_analysis_rnaseq

Analysis Setup:
We followed a reduced analysis workflow demonstrated by Balli using the SRA files for 8 samples: 2 normal, 3 mP, 3 mT). These samples represent extract to approximately 480 million reads and 150gb of data (FASTQ format). We perform this experiment first with 7 samples (2 normal, 3mP, 2mT) and then add the 8th sample to view the differences in the continuous analysis workflow.

We perform four preprocessing steps prior to beginning continuous analysis. We perform these steps prior to beginning continuous analysis to reduce the amount of time the process takes and limit necessary bandwidth. Author discretion should be used when performing time or resource consuming tasks following a common standard workflow that is unlikely to change.

1. Download the samples from the Sequence Read Archive (http://www.ncbi.nlm.nih.gov/sra?term=SRP049959 "SRR1654626", "SRR1654628", "SRR1654633", "SRR1654636", "SRR16546367", “SRR1654639”, "SRR1654637", "SRR1654641", "SRR1654643")
2. Split the .sra into fast q files using the SRA toolkit (http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=toolkit_doc)
3. Download the mouse reference genome assembly (http://hgdownload.soe.ucsc.edu/goldenPath/mm10/bigZips/refMrna.fa.gz)
4. We host the compressed RNA-Seq data (~90 GB gzipped FASTQ) on the cluster and access it via local FTP. Future versions of Drone will likely allow mounting of local drives for similar workflows.

Continuous Analysis Run:

1. Generate a kallisto index file from the reference file and quantify abundances of transcripts from each RNA-Seq sample (run on 28 cores)

```
- kallisto index -i mm10 refMrna.fa.gz
- kallisto quant -i mm10  -o kallisto_output/SRR1654626 --bootstrap-samples=10 SRR1654626_1.fastq.gz
    SRR1654626_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654628 --bootstrap-samples=10 SRR1654628_1.fastq.gz
    SRR1654628_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654633 --bootstrap-samples=10 SRR1654633_1.fastq.gz
    SRR1654633_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654636 --bootstrap-samples=10 SRR1654636_1.fastq.gz
    SRR1654636_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654637 --bootstrap-samples=10 SRR1654637_1.fastq.gz
    SRR1654637_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654639 --bootstrap-samples=10 SRR1654639_1.fastq.gz
    SRR1654639_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654641 --bootstrap-samples=10 SRR1654641_1.fastq.gz
    SRR1654641_2.fastq.gz -t 28
- kallisto quant -i mm10  -o kallisto_output/SRR1654643 --bootstrap-samples=10 SRR1654643_1.fastq.gz
    SRR1654643_2.fastq.gz -t 28
```

2. The next portion of the analysis is performed from ‘r_script.r’ and follows the workflow described by Balli:

    a. Generate the transcripts per million (TPM) matrix

```r
for (i in 1:length(kal_dirs)){
    print(kal_dirs[i])
    tmp = read.table(file = paste0(kal_dirs[i],"/abundance.tsv"), header = T)
    assign(kal_dirs[i], tmp)
}
sample_list = mget(kal_dirs)
ListNames <- Map(function(x, i) setNames(x, ifelse(names(x) %in% "target_id", names(x), sprintf("%s.%d", names(x), i))),
    sample_list, seq_along(sample_list))
# merge full kalisto table
# have to use Reduce() function as merge() will only merge two data.frames
full_results <- Reduce(function(...) merge(..., by='target_id', all = T), ListNames)

# subset estimate count values (correcting from TPM values)
count_vals <- full_results[, grep("est_counts", names(full_results))]

# now we have to assign column and row IDs to tpm table
row.names(count_vals) <- full_results$target_id

# setting column names is a little trickier as each file has "output_SRR..." as IDs
# we will use regrex with stringr and dplyr packages to get down to the SRR file IDs
splitNames <- str_split(string=kal_dirs, pattern = "_", n = 2)
df <- ldply(splitNames, data.frame)
FinalNames <- df[df$X..i.. != "output",]

FinalNames
# [1] SRR1654626 SRR1654628 SRR1654633 SRR1654636 SRR1654637 SRR1654639 SRR1654641 SRR1654643
# 9 Levels: output SRR1654626 SRR1654628 SRR1654633 SRR1654636 SRR1654637 ... SRR1654643
# actual IDs
ActualNames <- c("mN5","mN7","mP1","mP4","mP5","mT3","mT5","mT8")
Groups <- data.frame(Subtypes = c("mN","mN","mP","mP","mP","mT","mT","mT"))

Names_count <- data.frame(FinalNames, ActualNames, Groups)
Names_count

colnames(count_vals) <- ActualNames
head(count_vals)
```

    b. Create a matrix to specify the group each sample belongs to

```
des <- factor(Groups$Subtypes)
design <- model.matrix(~0+des)
colnames(design) <- levels(des)
design
```

c. Filter out lowly expressed genes

```
# filter out lowly expressed genes
A <- rowSums(count_vals)
isexpr <- A > 1
table(isexpr)
# isexpr
# FALSE TRUE
# 21367 66831
count_vals <- count_vals[isexpr, ]
dim(count_vals)
y <- DGEList(counts = count_vals)
```

d. Generate a principle component plot

```
pcaTpm = (pcaCount <- ggplot(scoresCount, aes(x=PC1, y=PC2)) +
 geom_point(size = 4, aes(col=factor(scoresCount$Subtypes))) +
 ggtitle("Principal Components\nUsing Kallisto Estimated Counts") +
 geom_text(aes(label=scoresCount$ActualNames),hjust=0.5, vjust=1) +
 theme_minimal())
ggsave(filename=file.path(base_dir, "results/PCA.png"), plot=pcaTpm, width=10, height=8)
```

e. Fit the limma linear model for differential gene expression analysis.

```
exp <- voom(counts = y$counts, design = design, plot = T)
fit <- lmFit(exp, design = design)

# set up contrast matrix of conditions
cont.matrix <- makeContrasts("mT-mN",'mT-mP','mN-mP', levels = design)
fit <- contrasts.fit(fit, cont.matrix)
fit <- eBayes(fit)
topTable(fit, adjust="BH", number = Inf, p=0.05)
summary(decideTests(fit))

# compare the mT vs mN and mP vs mN contrasts and sort by LogFC
options(digits = 3)

# comparing normal organoids to Tumor organoids
res_mTvsmN <- (topTable(fit, coef = 1, adjust = 'BH', n = Inf, p = 1)[,-2])
res_mTvsmN <- res_mTvsmN [order(-res_mTvsmN$logFC),]
head(res_mTvsmN)
nrow(res_mTvsmN)

res_mTvsmN$threshold = as.factor(abs(res_mTvsmN$logFC) > 2 & res_mTvsmN$P.Value < 0.05/(dim(res_mTvsmN)[1]))
```

f. Plot differential expression in the form of a volcano plot

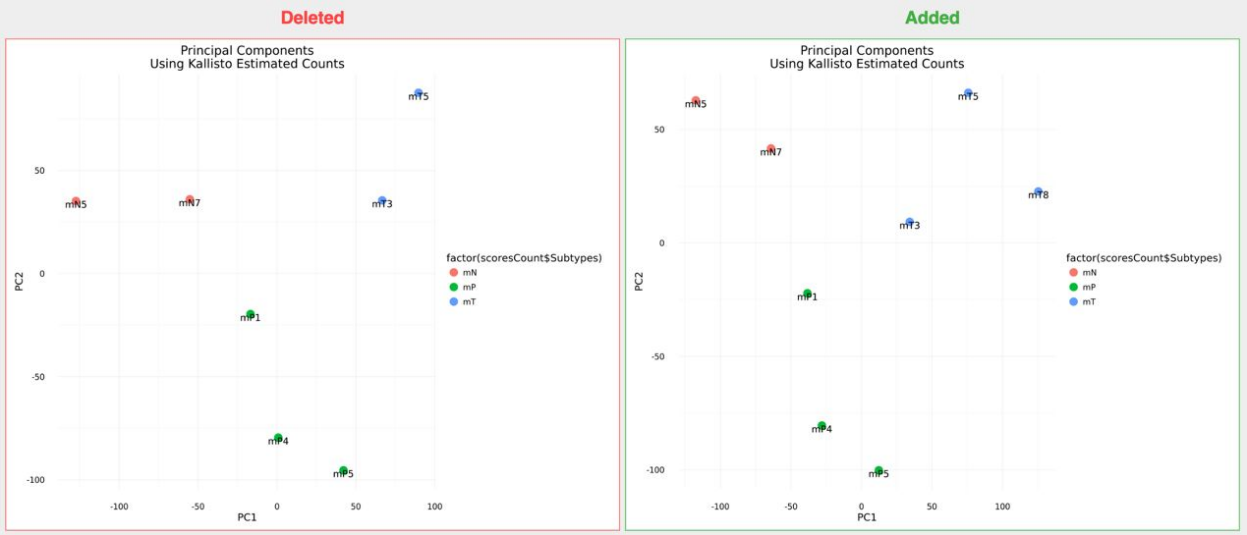
```
# Make a basic volcano plot
g = ggplot(data=res_mTvsmN, aes(x=logFC, y=-log10(P.Value), colour=threshold)) +
  geom_point(alpha=0.4, size=1.75) +
  xlim(c(-10, 10)) + ylim(c(0, 15)) +
  xlab("log2 fold change") + ylab("-log10 p-value")
ggsave(filename=file.path(base_dir, "results/res_mTvsmN.png"), plot=g, width=10, height=10)
```

3. Save static versions of the docker images before and after analysis. These can be uploaded to services like Zenodo and Figshare to receive DOIs.

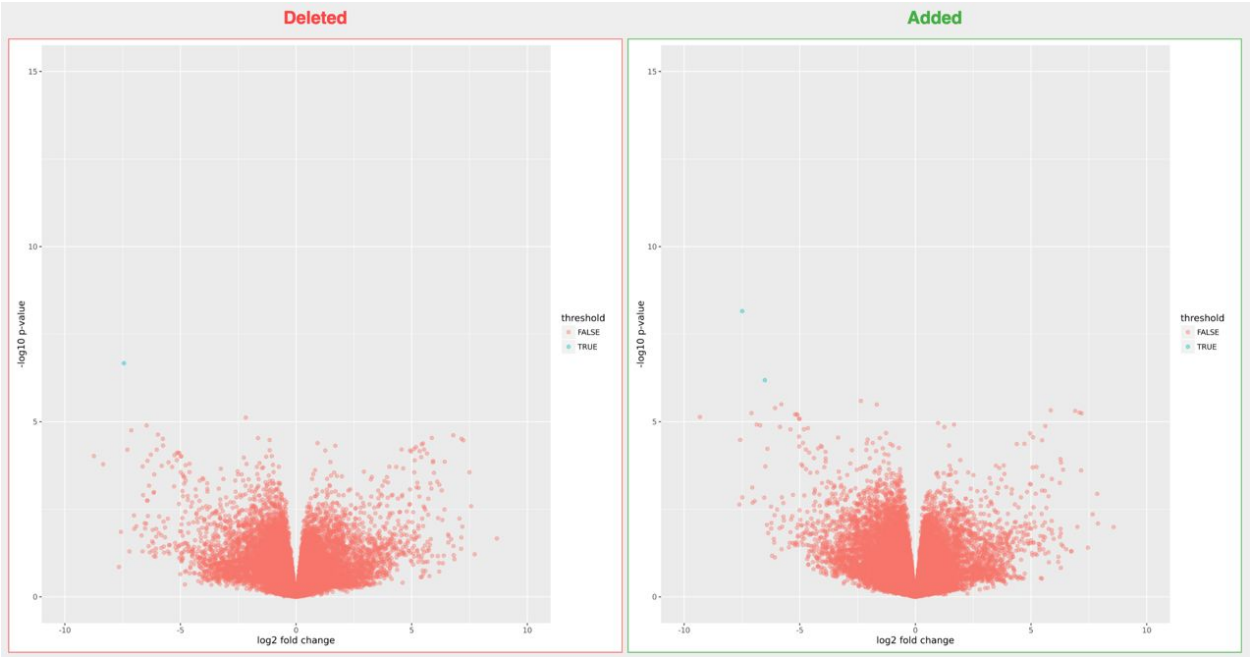
   docker save brettbj/continuous_analysis_rnaseq > continuous_analysis_rnaseq
   docker save brettbj/continuous_analysis_rnaseq_post > continuous_analysis_rnaseq_post

   https://dx.doi.org/10.6084/m9.figshare.3545156.v1

Sample Results:



Supplemental Figure 10. The effect of adding an additional organoid derived from pancreatic adenocarcinoma on principal components analysis using Kallisto's estimated counts.



Supplemental Figure 11. A volcano plot plotting the p-value vs. the log fold change. Adding an additional organoid derived from pancreatic adenocarcinoma leads to an additional gene being marked as significantly differentially expressed after Benjamini & Hochberg correction.

The docker image to rerun the analysis is available at:
https://hub.docker.com/r/brettbj/continuous_analysis_rna/
Static docker images are also available via Figshare.
https://dx.doi.org/10.6084/m9.figshare.3545156.v1

The addition of the eighth sample: https://git.io/v6kt6

The continuous analysis after the run: https://git.io/v6ktw

Continuous Analysis Configuration:

This example uses a local cluster setup to run the RNA-Seq processing, quantification and model fitting (limma and sleuth) in approximately 2 hours for 8 samples:

1. Install Docker on host machine - Linux, Mac, Windows
2. Pull the drone image via docker

    sudo docker pull drone/drone:0.4

3. Create a new application at - https://github.com/settings/developers - with your hosts ip address in the homepage URL and the authorization callback followed by /authorize/ (Supplemental Figure 6).

```
Homepage URL: http://YOUR-IP-HERE/
Callback URL: http://YOUR-IP-HERE/authorize/
```

4. Add a webhook in to notify the continuous integration server of any updates pushed to the repository (Supplemental Figure 7).
5. Create a configuration file on your local machine at (/etc/drone/dronerc), filling in the client information.

   REMOTE_DRIVER=github
   REMOTE_CONFIG=https://github.com?client_id=....&client_secret….

6. Access the drone control panel at your-ip-address/login and click the github button to login with github.
7. After logging in, choose the repository (continuous_analysis_rnaseq) you would like to use continuous analysis on.
8. In the settings tab of drone, increase the timeout to be 900 minutes from the default of 60 minutes.
9. Add a .drone.yml file to your repository and continuous integration will run with each new push to the repository.

   https://github.com/greenelab/continuous_analysis_rnaseq/blob/master/.drone.yml

   In the Kallisto quantification step we specify to use 28 threads. This can be extended to the number of cores available