# rPlant

Barb Banbury, University of Tennessee, bbanbury@utk.edu
Kurt Michels, University of Arizona, kamichels@math.arizona.edu

August 27, 2013

## Contents

**1 Introduction 2**

**2 Getting Started 2**
2.1 Validation of users . . . . . . . . . . 3

**3 Uploading Files 3**
3.1 UploadFile function . . . . . . . . . . 3
3.2 Supported File Types . . . . . . . . . . 4

**4 Manipulating directories on iPlant servers 4**
4.1 Listing directories . . . . . . . . . . 4
4.2 Making Directories . . . . . . . . . . 5
4.3 Sharing Directories . . . . . . . . . . 5
4.4 Renaming Directories . . . . . . . . . . 6
4.5 Moving Directories . . . . . . . . . . 6
4.6 Deleting Directories . . . . . . . . . . 6

**5 Manipulating files on iPlant servers 7**
5.1 Sharing Files . . . . . . . . . . 7
5.2 Moving Files . . . . . . . . . . 7
5.3 Renaming Files . . . . . . . . . . 7
5.4 Deleting Files . . . . . . . . . . 8

**6 Applications in the rPlant package 8**
6.1 Listing Applications . . . . . . . . . . 8
6.2 Individual application information . . . . . . . . . . 10

**7 Submitting Jobs in the rPlant package 11**
7.1 Submitting Job . . . . . . . . . . 11
7.2 Submitting a job with a shared file . . . . . . . . . . 12

**8 Checking Job Status and Retreiving Job output 12**
8.1 Checking job status . . . . . . . . . . 13
8.2 Listing job status . . . . . . . . . . 13
8.3 Looking at Job History . . . . . . . . . . 14
8.4 Retrieve job files . . . . . . . . . . 14
8.5 Delete job . . . . . . . . . . 14

**9 Advanced job submittal 15**
9.1 Submitting a job with the wrappers . . . . . . . . . . . . . . . . . . . . . 15
9.2 Muscle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
9.3 Mafft . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
9.4 ClustalW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
9.5 FastTree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
9.6 RAxML (Randomized Accelerated Maximum Likelihood) . . . . . . . . . . . 20
9.7 PHYLIP-Parsimony 3.69 . . . . . . . . . . . . . . . . . . . . . . . . . . 20
9.8 Genome Wide Association Study models . . . . . . . . . . . . . . . . . . 21
9.9 PLINK . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
9.10 PLINK Conversion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
9.11 FaST-LMM (Factored Spectrally Transformed Linear Mixed Models) . . . . 22

**10 Creating workflows 22**
10.1 Workflow One . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
10.2 Workflow Two . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
10.3 Workflow Three . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
10.4 Workflow Four . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

## 1 Introduction

The iPlant Collaborative has developed many resources to deal with the emerging computational challenges facing biology. The project was initially designed to support the plant sciences, but thanks to a generic approach, can be equally used by other disciplines. Users have access to many different applications for data analysis, including clustering/network analyses, QTL mapping, sequence alignments, phylogenetic tree building, and comparative methods.

The main interface is its user-friendly Discovery Environment (`http://www.iplantcollaborative.org/discover/discovery-environment`). The second interface, which is linked to the Discovery Environment, is called the Foundation API. The Foundation API is used for the more computationally intensive applications. The API is a RESTful application programming interface (API; Fielding 2000) that allows direct interaction with all of iPlant resources. The only way to access the API is to use '`curl`' statements (cite), an example of a `curl` statement will be detailed in one of the sections. The API provides access to authentication, data manipulation and storage, and job submittal via HTTPS- and command-line functions (`https://foundation.iplantcollaborative.org`). The benefit of using the API is having programmatic access that allows advantages to power users (e.g. submitting jobs via batch files). The *rPlant* package provides a direct link between high performance resources located at the Texas Advanced Computing Center (`http://www.tacc.utexas.edu`) that the API can access and the R environment, by essentially creating wrappers around the `curl` statements.

## 2 Getting Started

This vignette assumes you have the current version of R. First, install and load the package. A stable release is available through CRAN (`http://cran.r-project.org/web/packages/rPlant/`) or a working repository can also be used through R-Forge (`https://r-forge.`

`r-project.org/projects/rplant/`).

You can register as an iPlant user on their website (`http://user.iplantcollaborative.org/`) generating a unique username and password combination.

## 2.1 Validation of users

`Validate(user, pwd, api="iplant", print.curl=FALSE)`

This username/password combination will be used in the `validate` function. The `validate` function is required for EVERY rPlant session. It is the first thing that must be executed or else the rest of the session will not work.

```
> require(rPlant)
> user.name <- "enter your username"
> user.pwd <- "enter your secret password"
> Validate(user.name, user.pwd)
```

```
[1] "Authentication failed"
```

The function checks if the username and password are valid iPlant credentials. If they aren't the above error is displayed. If the function is successful then nothing is printed.

```
> Validate(user.name, user.pwd)
```

*Note*: This package abides by the unix rule, ”silence is golden”. If a function is successful then no output will be displayed. If an error is attained then the error will be printed.

```
> Validate(user.name, user.pwd, print.curl=TRUE)
```

```
[1] "curl -sku 'henryl' https://foundation.iplantc.org/auth-v1/"
```

*Note*: Every *rPlant* function has the option '`print.curl=TRUE`' or '`FALSE`'. This refers to `cURL` a computer software project providing a way to transfer data using various protocols, for detail on `cURL` see `http://en.wikipedia.org/wiki/CURL`. These statements (w/o the outside quotes) can be copied and pasted into a terminal in linux or unix. And if `cURL` is installed on the computer then the statements can be executed. You will see that these statements do the exact same thing as the *rPlant* function. This is one of the big advantages of *rPlant*, it can be used on any computer (including windows) and there is no need for the user to install `cURL` on that computer, because *rPlant* uses the package `RCurl`.

# 3 Uploading Files

## 3.1 UploadFile function

`UploadFile(local.file.name, local.file.path="", filetype=NULL, print.curl=FALSE, suppress.Warnings=FALSE)`

The first step is to upload files onto iPlants. The biggest confusion I believe is that the Upload file does NOT take a file from the R workspace and upload it onto iPlants servers, instead it takes a file from your computer and uploads it onto iPlants servers.

```
> data(DNA.fasta)
> write.fasta(sequences = DNA.fasta, names = names(DNA.fasta), file.out = "DNA.fasta")
> UploadFile(local.file.name="DNA.fasta", filetype="FASTA-0")
```

An error that can be recorded is if the file "DNA.fasta" was already on the iPlant server. If it was the error would be returned and the file not uploaded. For the fasta file the file type is "`FASTA-0`".

## 3.2 Supported File Types

`SupportFile(print.curl=FALSE)`

There are 33 other file types supported by iPlant, use the `SupportFile` function to see all of the available file types, i.e. `PHYLIP` file type is "`PHYLIP-0`" and `ClustalW` is "`ClustalW-1.8`".

```
> SupportFile()
```

```
 [1] "2bit-0"            "ASN-0"           "BAM-0.1.2"       "Barcode-0"
 [5] "BED-0"             "BlastN-2.0"      "Bowtie-0"        "BZIP2-0"
 [9] "CEL-3"             "ClustalW-1.8"    "CSV-0"           "DOT-0"
[13] "EMBL-0"            "EXPR-0"          "FAI-0"           "FASTA-0"
[17] "FASTQ-Illumina-0" "FASTQ-Int-0"     "FASTQ-Solexa-0"  "FASTQ-0"
[21] "Genbank-0"         "GFF-2.0"         "GFF-3.0"         "GFF-3.0"
[25] "GraphML-0"         "GTF-2.2"         "HTML-4"          "HTML-5"
[29] "Newick-0"          "NEXUS-0"         "PAIR-0"          "PDB-3.2"
[33] "Phylip-0"          "PhyloXML-1.10"   "Pileup-0"        "SAI-0.1.2"
[37] "SAM-0.1.2"         "SBML-1.2"        "SBML-2.4.1"      "SBML-3.1"
[41] "Soap-PE-1"         "Soap-SE-1"       "Stockholm-1.0"   "TAB-0"
[45] "TAR-0"             "Text-0"          "VCF-3.3"         "VCF-4.0"
[49] "WIG-0"
```

# 4 Manipulating directories on iPlant servers

Now that the file "DNA.fasta" has been uploaded onto the iPlant servers we can look at the file (or at least see which directory the file is in) by using the `ListDir` function. There are a few other directory manipulation functions, they are: `MakeDir`, `ShareDir`, `RenameDir`, `MoveDir` and `DeleteDir`.

## 4.1 Listing directories

```
ListDir(dir.name, dir.path="", print.curl=FALSE, shared.username=NULL,
suppress.Warnings=FALSE)
```

Looking in the home directory you can see the "DNA.fasta" file.

```
> ListDir(dir.name="", suppress.Warnings=TRUE)
```

```
     name        type
[1,] ".."        "dir"
[2,] "DNA.fasta" "file"
```

*Note*: Some functions contain an option, "`suppress.Warnings`". If you are absolutely sure that the commands you are entering are correct then to speed up the process have `suppress.Warnings=TRUE`. But be careful, if used inappropriately then files could get overwritten or the files might not even exist and you will get no warning about it.

## 4.2 Making Directories

```
MakeDir(dir.name, dir.path="", print.curl=FALSE, suppress.Warnings=FALSE)
```

The following function is very self explanatory, it will make a directory '`hello`' in the home directory.

```
> MakeDir(dir.name="hello")
```

Again making the directoy '`all`' in the '`hello` directory.

```
> MakeDir(dir.name="all", dir.path="hello")
```

I'm making another directory '`robots`' in the '`all` directory. I'm showing this so you can see how the `dir.path` needs to be constructed, and how the `dir.name` and `dir.path` are related. All of the functions have this same format.

```
> MakeDir(dir.name="robots", dir.path="hello/all")
```

We can look inside the "`hello/all/robots`" directory and see that there is nothing in there.

```
> ListDir(dir.name="robots", dir.path="hello/all")
```

```
     name type
[1,] ".." "dir"
```

## 4.3 Sharing Directories

```
ShareDir(dir.name, dir.path="", shared.username, read=TRUE, execute=TRUE,
print.curl=FALSE, suppress.Warnings=FALSE)
```

A really nice feature of *iPlant* is the file sharing feature. As was said in the introduction one of *iPlant*'s goals was to work with very large data sets. And when data sets are too large to send via e.mail then a sharing feature is absolutely necessary. There are in fact two `Share` functions, on for sharing a single file and the other (this one) for sharing an entire directory.

In this sample we share the `all` directory. Notice that all subdirectories of the directory `all` will be shared.

```
> ShareDir(dir.name="all", dir.path="hello", shared.username="phyllisl")
```

Now, in the above example I share something with '`phyllisl`'. I'm going to make a switch now I'm going to look at a folder that `phyllisl` had shared with me.

```
> ListDir(dir.name="data", dir.path="", shared.username="phyllisl")
```

```
     name            type
[1,] ".."            "dir"
[2,] "muscle3.fasta" "file"
```

There are other functions where the `shared.username` is used. It is with the `SubmitJob` function and the wrappers.

## 4.4 Renaming Directories

```
RenameDir(dir.name, new.dir.name, dir.path="", print.curl=FALSE,
suppress.Warnings=FALSE)
```

This function is self explanatory, it renames a directory.

```
> RenameDir("robots", "tools", "hello/all")
```

That command can be verified.

```
> ListDir("all", "hello")
```

```
     name    type
[1,] ".."    "dir"
[2,] "tools" "dir"
```

And you can see that it has been changed.

*Note*: When the directory is renamed it is no longer shared.

## 4.5 Moving Directories

```
MoveDir(dir.name, dir.path="", end.path="", print.curl=FALSE,
suppress.Warnings=FALSE)
```

This function is self explanatory, it moves a directory.

```
> MoveDir("tools", "hello/all", end.path="")
```

The move took the directory `tools` from `hello/all` to the home directory. Verified below.

```
> ListDir("")
```

```
     name        type
[1,] ".."        "dir"
[2,] "hello"     "dir"
[3,] "tools"     "dir"
[4,] "DNA.fasta" "file"
```

And you can see that it has been changed.

*Note*: When the directory is moved it is no longer shared.

## 4.6 Deleting Directories

```
DeleteDir(dir.name, dir.path="", print.curl=FALSE, suppress.Warnings=FALSE)
```

This function is self explanatory, it deletes a directory and all of the subdirectories.

```
> DeleteDir("tools")
```

Verified below.

```
> ListDir("")
```

```
     name        type
[1,] ".."        "dir"
[2,] "hello"     "dir"
[3,] "DNA.fasta" "file"
```

The directory `tools` is no longer in the home directory.

*Note*: Clearly when the directory is deleted it is no longer shared.

## 5 Manipulating files on iPlant servers

The file manipulation tools available in this package are very similar to the directory manipulation tools. The file manipulation functions are: `ShareFile`, `RenameFile`, `MoveFile` and `DeleteFile`.

### 5.1 Sharing Files

```
ShareFile(file.name, file.path="", shared.username, read=TRUE, execute=TRUE,
print.curl=FALSE, suppress.Warnings=FALSE)
```

As described in the `ShareDir` function a really nice feature of *iPlant* is the file sharing feature. This is the “other” file-sharing funtion, and it just shares one file at a time.

Now I'm going to share the file “`DNA.fasta`” with “`phyllisl`”

```
> ShareFile(file.name="DNA.fasta", shared.username="phyllisl")
```

### 5.2 Moving Files

```
MoveFile(file.name, file.path="", end.path="", print.curl=FALSE,
suppress.Warnings=FALSE)
```

This function is self explanatory, it moves the file.

```
> MoveFile("DNA.fasta", end.path="hello/all")
```

The move took the file “`DNA.fasta`” from the home directory into the `hello/all` directory. Verified below.

```
> ListDir("all", "hello")
```

```
     name        type
[1,] ".."        "dir"
[2,] "DNA.fasta" "file"
```

And you can see that it has been changed.

*Note*: When the file is moved it is no longer shared.

### 5.3 Renaming Files

```
RenameFile(file.name, new.file.name, file.path="", print.curl=FALSE,
suppress.Warnings=FALSE)
```

This function is self explanatory, it renames a file.

```
> RenameFile("DNA.fasta", "lp.fasta", "hello/all")
```

That command can be verified.

```
> ListDir("all", "hello")

     name       type
[1,] ".."       "dir"
[2,] "lp.fasta" "file"
```

And you can see that it has been changed.

*Note*: When the file is renamed it is no longer shared.

## 5.4 Deleting Files

`DeleteFile(file.name, file.path="", print.curl=FALSE, suppress.Warnings=FALSE)`

This function is self explanatory, it deletes a file in the specified directory.

```
> DeleteFile("lp.fasta", "hello/all")
```

Verified below.

```
> ListDir("all", "hello")

     name type
[1,] ".." "dir"
```

The file "`lp.fasta`" is no longer in the `hello/all` directory.

*Note*: Clearly when the file is deleted it is no longer shared.

# 6 Applications in the rPlant package

The real power in the *rPlant* package is to have dozens of phylogenetic tools/applications at your finger tips. *rPlant* can be used to interact with any of the applications available via the API.

## 6.1 Listing Applications

`ListApps(description=FALSE, print.curl=FALSE)`

This aptly named function returns a sorted list of the newest versions of the public applications that are available via the Foundation API. These applications are ones that can be used in the `SubmitJob` function.

```
> ListApps(description=TRUE)

 [1] "abyss-lonestar-1.3.3u1 - ABySS is a de novo, parallel, paired-end sequence assembler"
 [2] "abyss-lonestar-1.3.4u1 - ABySS is a de novo, parallel, paired-end sequence assembler"
 [3] "AllpathsLG_lonestar-44837u1 - AllpathsLG, genome assembler"
 [4] "autodock_vina-1.00u1 - AutoDock Vina is a new open-source program for drug discovery, molecular dock
 [5] "bismark-0.7.4u1 - Bismark is a program to map bisulfite treated sequencing reads to a genome of inte
 [6] "bismark_genome_preparation-0.7.4u1 - genome preparation for bismark"
 [7] "bismark_methylation_extractor-0.7.4u1 - Extracting methylation in 3 contexts from bismark result"
 [8] "blastx-stampede-ncbi-db-2.2.26u2 - Intended for metagenome analysis or post-assembly contig annotati
 [9] "bwa-lonestar-0.5.9u3 - bwa 0.5.9 is a next gen sequence aligner"
[10] "ClustalW2-2.1u1 - Multiple alignment of nucleic acid and protein sequences"
[11] "clustalw2Dispatcher-1.0.13100u1 - Multiple alignment of nucleic acid and protein sequences"
```

```
[12] "clustalw2-lonestar-2.1u2 - Multiple alignment of nucleic acid and protein sequences"
[13] "dnalc-cuffdiff-lonestar-2.1.1u3 - Find significant changes in transcript expression, splicing, and pr
[14] "dnalc-cuffdiff-stampede-2.1.1u3 - Find significant changes in transcript expression, splicing, and pr
[15] "dnalc-cufflinks-lonestar-2.1.1u2 - Transcript assembly and basic quantitation for RNA-Seq"
[16] "dnalc-cufflinks-stampede-2.1.1u2 - Transcript assembly and basic quantitation for RNA-Seq"
[17] "dnalc-cuffmerge-lonestar-2.1.1u1 - Transcript assembly and merge for RNA-Seq data"
[18] "dnalc-cuffmerge-stampede-2.1.1u1 - Transcript assembly and merge for RNA-Seq data"
[19] "dnalc-fastqc-lonestar-0.10.1u1 - "
[20] "dnalc-fastqc-stampede-0.10.1u1 - "
[21] "dnalc-fastx-lonestar-0.0.13.2u1 - FASTQ/A short-reads pre-processing tools"
[22] "dnalc-fastx-stampede-0.0.13.2u2 - FASTQ/A short-reads pre-processing tools"
[23] "dnalc-fxtrim-lonestar-0.0.13.2u1 - FASTQ/A short-reads pre-processing tools"
[24] "dnalc-fxtrim-stampede-0.0.13.2u1 - FASTQ/A short-reads pre-processing tools"
[25] "dnalc-tophat-lonestar-2.0.8u1 - A spliced read mapper for RNA-Seq"
[26] "dnalc-tophat-stampede-2.0.8u2 - A spliced read mapper for RNA-Seq"
[27] "FaST-LMM-1.09u1 - FaST-LMM (Factored Spectrally Transformed Linear Mixed Models) is a program for per
[28] "fasttreeDispatcher-1.0.0u1 - FastTree infers approximately-maximum-likelihood phylogenetic trees from
[29] "forward-regression-0.0.1u1 - FR: Partitioned Linear Model based Forward Regression"
[30] "gapcloser-1.12u1 - Extra module with Soapdenovo2"
[31] "gatk-1000bulls-geno-lonestar-1.00u1 - "
[32] "GeneSeqer-5.0u1 - GeneSeqer, a parallel (MPI), gapped mapper for ESTs"
[33] "GMAP_stampede-121212u1 - GMAP, a multithreaded, gapped mapper for ESTs"
[34] "GSNAP_lonestar-121212u1 - GSNAP, a multithreaded, gapped mapper for ESTs"
[35] "GSNAP_stampede-121212u2 - GSNAP, a multithreaded, gapped mapper for ESTs"
[36] "head-stampede-5.97u2 - This is an application you can use to inspect the beginning of a file."
[37] "head-trestles-5.97u1 - This is an application you can use to inspect the beginning of a file."
[38] "mafftDispatcher-1.0.13100u1 - MAFFT is a multiple sequence alignment program for unix-like operating
[39] "mafft-lonestar-6.864u1 - MAFFT is a multiple sequence alignment program for unix-like operating syste
[40] "metagenemark-1.00u3 - An algorithm for accurate ab initio gene prediction in DNA sequences derived fr
[41] "metaphlan-lonestar-1.6.0u4 - MetaPhlAn is a computational tool for profiling the composition of micro
[42] "Muscle-3.8.31u1 - MUSCLE is a program for creating multiple alignments of amino acid or nucleotide se
[43] "Muscle-3.8.32u4 - MUSCLE is a program for creating multiple alignments of amino acid or nucleotide se
[44] "muscle-lonestar-3.8.31u2 - MUSCLE is a program for creating multiple alignments of amino acid or nucl
[45] "newbler-2.6.0u1 - Genome assembler for 454 sequencing reads"
[46] "oases-0.2.08u1 - Transcript assembler for short sequencing reads, works with Velvet."
[47] "phylip-dna-parsimony-lonestar-3.69u2 - Estimates phylogenies by the parsimony method using nucleic ac
[48] "phylip-protein-parsimony-lonestar-3.69u2 - Estimates phylogenies by the parsimony method using amino
[49] "plink-1.07u1 - Open-source whole genome association analysis toolset designed to perform a range of l
[50] "quicktree-dm-lonestar-1.1u2 - Reconstruction of phylogenies for very large protein families that woul
[51] "quicktree-tree-lonestar-1.1u2 - Reconstruction of phylogenies for very large protein families that wo
[52] "quline-lonestar-3-0.11u1 - QU-GENE does simulations"
[53] "raxml-lonestar-7.2.8u1 - RAxML is a program for sequential and parallel Maximum Likelihood based infe
[54] "ray-2.2.0u1 - Genome assembler for short sequencing reads."
[55] "scarf-1.00u1 - A next-gen sequence assembly tool for evolutionary genomics. Designed especially for a
[56] "soapdenovo-1.05u1 - Genome assembler for Illumina sequencing reads"
[57] "soapdenovo-2.04u1 - Genome assembler for Illumina sequencing reads"
[58] "STRUCTURE-2.3.4u1 - population structure"
[59] "TASSEL4-GLM-0.0.1u1 - General Linear Model"
[60] "TASSEL4-MLM-0.0.1u1 - Mixed Linear Model"
[61] "trinity_lonestar4-20121005u1 - Trinity represents a novel method for the efficient and robust de novo
[62] "trinity_r2012-03-17_lonestar4-1.00u1 - Trinity represents a novel method for the efficient and robust
[63] "velvetg-1.2.07u2 - Genome assembler for short sequencing reads, second stage."
[64] "velveth-1.2.07u1 - Genome assembler for short sequencing reads, first stage."
[65] "wc-1.00u1 - Count words in a file"
```

*Note*: As said in the prior paragraph these applications listed are PUBLIC applications. Applications in the Foundation API are split into two categories, public and private. Private applications are ones that are developed and tested and changed. Only the user who created the private application can use it. The other category is public applications. After a private application has gone through extensive testing, then the application can be published and it becomes a public application which is available to all iPlant users. In the Foundation API a public application is labeled by adding the suffix "u1" to it. The '1' is referred to as the version number, so if a public application is fixed and republished the suffix becomes "u2".

## 6.2 Individual application information

`GetAppInfo(application, return.json=FALSE, print.curl=FALSE)`

The `ListApps` function only lists the applications and a short description. To get more detailed information for each application use the `GetAppInfo` function.

```
> GetAppInfo("velveth-1.2.07u2")

$Description
[1] "Genome assembler for short sequencing reads, first stage."

$Application
[1] "velveth-1.2.07u2" "Public App"       "Newest Version"

$Information
      kind         id               fileType/value
 [1,] "input"      "reads5"         "fasta-0"
 [2,] "input"      "reads6"         "fasta-0"
 [3,] "input"      "reads2"         "fasta-0"
 [4,] "input"      "reads4"         "fasta-0"
 [5,] "input"      "reads1"         "fasta-0"
 [6,] "input"      "reads3"         "fasta-0"
 [7,] "parameters" "strandSpecific" "string"
 [8,] "parameters" "Output"         "string"
 [9,] "parameters" "kmer"           "string"
[10,] "parameters" "format5"        "string"
[11,] "parameters" "format2"        "string"
[12,] "parameters" "format1"        "string"
[13,] "parameters" "format3"        "string"
[14,] "parameters" "format4"        "string"
      details
 [1,] "Sequence file in fastq, fasta, or sam format"
 [2,] "Reference sequence file in fasta format"
 [3,] "Sequence file in fastq, fasta, or sam format"
 [4,] "Sequence file in fastq, fasta, or sam format"
 [5,] "Sequence file in fastq, fasta, or sam format"
 [6,] "Sequence file in fastq, fasta, or sam format"
 [7,] "strand specific"
 [8,] "Name for output directory"
 [9,] "kmer size"
[10,] "sequence file format, library 5"
[11,] "sequence file format, library 2"
[12,] "sequence file format, library 1"
[13,] "sequence file format, library 3"
[14,] "sequence file format, library 4"
```

The `GetAppInfo` function gives critical information on the `application` that is needed in the `SubmitJob` function. A list of information is outputted. The first element gives a short description of the application. The second element in the list gives basic information on the application including is it a public application and if it is the newest version. Both are important information. If the application is a private application it can only be run by the person who submitted the application to the Foundation API, and clearly you want to run the newest version of the public application.

The third element in the list the matrix outputted gives four columns of information. The first column, labeled 'kind, tells the "input", sometimes the "output" and "parameters" from the application. The second column, labeled id, give the name of the "input", etc. For example, GetAppInfo("velveth-1.2.07u2")[[2]], the 'kind' column states there are six inputs for this app, and the 'id' column the names of those inputs are "reads5","reads3", etc. There are also eight parameters for the app, the paramters are format2, etc. The third column in the matrix is 'fileType/value. For the input this tells the file type which is important because if the wrong fileType is inputted into the function, then the function will not work. For the parameters the third column contains the type of input necessary for the parameters, common ones are string, boolean, etc. The last column gives brief details on each input.

# 7 Submitting Jobs in the rPlant package

## 7.1 Submitting Job

```
SubmitJob(application, file.path="", file.list=NULL, input.list, args.list=NULL,
job.name, nprocs=1, private.APP=FALSE, suppress.Warnings=FALSE, shared.username=NULL,
print.curl=FALSE)
```

An important benefit of using rPlant is the ability to create batch-scripted files that automate job submittal and retrieval. For example, a user could submit parallel alignment jobs of different gene regions or multiple jobs with the same data and different parameter values. The results could then be automatically downloaded upon completion.

The following job submittal is a standard job submittal and there are a few things to take note of. *Sidenote: I am using this example to highlight some of the options available in this function, it might not be a true working example.*

```
> myJobM <- SubmitJob(application="Muscle-3.8.32u4", file.list=list("DNA.fasta"),
+                     input.list=list("stdin"), args.list=list(c("arguments",
+                     "-phyiout -center -cluster1 upgma")), job.name="Muscle")

Job submitted. You can check your job using CheckJobStatus(25916)
```

*Note 1*: input.list: From the GetAppInfo("Muscle-3.8.32u4") function, from the second list (GetAppInfo("Muscle-3.8.32u4")[[2]], the 'kind' column states there is one input for this app, and the 'id' column names that input as 'stdin'. This input type changes from application to application.

*Note 2*: file.list: Very related to the input.list, the file.list is the same length as the input.list. Also from the second list on the GetAppInfo function (GetAppInfo("Muscle-3.8.32u4")[[2]], the 'fileType/value' column says "FASTA-0". This indicates that the file in file.list must have the FASTA file type. If it the file types don't match then the application will fail.

*Note 3*: args.list: The args.list is where extra flags that can change default options can be entered. Again using the second list on the GetAppInfo function (GetAppInfo("Muscle-

3.8.32u4")[[2]], the ‘kind’ column states there is one parameters for this app, the ‘id’ column tells me the name of that parameter is ‘arguments’, and the ‘fileType/value’ column tells me it is a string. This is where the fourth column (‘details’) comes in handy; it tells me that the parameter input is ‘program arguments and options’. So what that tells me is that this string acts like a command line argument, any flags that you want changed, add them to this parameter. Now, the way that this information is used is in the following way:

```
args.list=list(c(arguments, "-phyiout -center -cluster1 upgma"))
```

The args.list is a list that is as long as the number of parameters (so length 1 in this example), so that means there as many vectors (c()) as there are parameters. All vectors c(arguments, "-center -cluster1 upgma") are of the same length, two. In position one is the name of the parameter, arguments, and in position two is the value of that parameter, “-phyiout -center -cluster1 upgma”, a string of command line arguments in this example.

*Note 4*: myJobM: It can be seen a list of length two is outputted to myJobM. In myJobM[[1]] is the job number and in myJobM[[2]] is the job name, both are very important information.

*Note 5*: If the SubmitJob is succesful, then the function automatically creates the folder “analyses”. Now if the job finishes, then a folder is created in the analyses folder, it is named after the job name. So, as in *Note 4*, that names is stored in myJobM[[2]].

## 7.2 Submitting a job with a shared file

Remember the file that had been shared with my by phyllisl? I am able to submit a job using that file.

```
> myJobS <- SubmitJob(application="Muscle-3.8.32u4", file.list=list("muscle3.fasta"),
+                     file.path="data", shared.username="phyllisl",
+                     args.list=list(c("arguments", "-fastaout")),
+                     input.list=list("stdin"), job.name="MuscleShare")

Job submitted. You can check your job using CheckJobStatus(25917)
```

# 8 Checking Job Status and Retreiving Job output

Once the job is submitted, you noticed how I stored something in myJobS.

```
> myJobS

[[1]]
[1] 25917

[[2]]
[1] "MuscleShare_2013-08-26_18-12-25.454"
```

It can be seen that there are two things outputted to myJobS, the job number and the job name, both are very important information. Using the job number a variety of options are available: CheckJobStatus, ListJobOutput, RetrieveJob and DeleteJob.

## 8.1 Checking job status

```
CheckJobStatus(job.id, print.curl=FALSE)
```

This is a very self-explanatory function, it checks the job status on the iPlant servers.

```
> CheckJobStatus(myJobS[[1]])
[1] "PENDING"
```

Table 1: Possible Outputs for `CheckJobStatus`

| **Stages** |
|---|
| PENDING |
| STAGING_INPUTS |
| CLEANING_UP |
| ARCHIVING |
| STAGING_JOB |
| FINISHED |
| KILLED |
| FAILED |
| STOPPED |
| RUNNING |
| PAUSED |
| QUEUED |
| SUBMITTING |
| STAGED |
| PROCESSING_INPUTS |
| ARCHIVING_FINISHED |
| ARCHIVING_FAILED |

## 8.2 Listing job status

```
ListJobOutput(job.id, print.curl=FALSE, print.total=TRUE)
```

Again this is a very self-explanatory function, once the job is finished then it lists the output files from the job.

```
> print(ListJobOutput(myJobS[[1]]))
[1] "Job is PENDING"
```

Notice that an error was shown, obviously the job output can't be found until the job is finished.

Now . . . waiting

```
> ListJobOutput(myJobS[[1]])
[1] "There are 4 output files for job '25917'"
[1] "fasta.aln"
[2] "muscle3.fasta"
[3] "muscleshare_2013-08-26_18-12-25454-25917.err"
[4] "muscleshare_2013-08-26_18-12-25454-25917.out"
```

These are all of the output files that are created for the `MUSCLE` job.

## 8.3 Looking at Job History

```
GetJobHistory(return.json=FALSE, print.curl=FALSE)
```

Again this is a very self-explanatory function, but for this function no job id is required. The reason is because this function displays the ENTIRE job history, not just one.

```
> GetJobHistory()
```

```
    job.id  job.name                               application
job "25917" "MuscleShare_2013-08-26_18-12-25.454" "Muscle-3.8.32u4"
job "25916" "Muscle_2013-08-26_18-12-18.488"      "Muscle-3.8.32u4"
job "25914" "PCout_2013-08-26_18-00-57.605"       "plink-1.07u1"
job "25913" "PCout_2013-08-26_17-54-31.236"       "plink-1.07u1"
job "25912" "PLINK_2013-08-26_17-51-48.019"       "plink-1.07u1"
job "25910" "geno_test_2013-08-26_17-43-01.043"   "FaST-LMM-1.09u1"
job "25911" "FaST-LMM_2013-08-26_17-43-38.901"    "FaST-LMM-1.09u1"
    status
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
```

## 8.4 Retrieve job files

```
RetrieveJob(job.id, file.vec, print.curl=FALSE, verbose=FALSE)
```

This is another really nice thing about the *rPlant* package. The ability to download the files directly from the iPlant servers to your computer. The following downloads all of the output files.

```
> RetrieveJob(myJobS[[1]], file.vec=ListJobOutput(myJobS[[1]], print.total=FALSE))
```

The files have been downloaded into the folder `MuscleShare_2013-08-26_18-12-25.454` (which is the job name), in the directory `/home/michels/Desktop/rPlant_vignette`. If you want to download just one or two of the files, do the following:

```
> RetrieveJob(myJobS[[1]], file.vec=c("fasta.aln"))
```

## 8.5 Delete job

```
DeleteJob(job.id, print.curl=FALSE, ALL=FALSE)
```

After the job has been submitted and the result files downloaded and you have no need for the job anymore one can use the `DeleteJob` function to clearly, delete the job. The nice thing about this function is that not only will it delete the job number from the job history but it will delete the job folder (in the `analyses` folder, see *Note 5* under `SubmitJob`) as well.

```
> DeleteJob(myJobS[[1]])
```

Proof that the job is deleted

```
> GetJobHistory()
```

```
    job.id  job.name                            application
job "25916" "Muscle_2013-08-26_18-12-18.488"    "Muscle-3.8.32u4"
job "25914" "PCout_2013-08-26_18-00-57.605"     "plink-1.07u1"
job "25913" "PCout_2013-08-26_17-54-31.236"     "plink-1.07u1"
job "25912" "PLINK_2013-08-26_17-51-48.019"     "plink-1.07u1"
job "25910" "geno_test_2013-08-26_17-43-01.043" "FaST-LMM-1.09u1"
job "25911" "FaST-LMM_2013-08-26_17-43-38.901"  "FaST-LMM-1.09u1"
    status
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
job "ARCHIVING_FINISHED"
```

You can see that the job is gone. Now another nice thing about this function is that if there are a lot of jobs in your job history because you haven't done a good job of keeping it clean, then execute the following:

```
> DeleteJob(ALL=TRUE)
```

Proof that the jobs and the corresponding folders are gone:

```
> GetJobHistory()
```

```
[1] "No jobs in history"
```

and . . .

```
> ListDir("analyses")
```

```
     name type
[1,] ".." "dir"
```

# 9 Advanced job submittal

## 9.1 Submitting a job with the wrappers

Currently, we have ten dedicated application wrappers for the 33+ programs available. Of the programs that have wrappers there is a clear bias towards phylogenetic applications, because the authors are evolutionary biologists. Writing wrapper functions is not programmatically difficult, but it does require familiarity with the individual programs and their associated data sets. We would like to encourage any users who are using programs without wrappers to submit patches adding wrapper functions or request to be a developer.

Among the wrappers there are three which do alignments: `Muscle`, `Mafft` and `ClustalW`. The alignments will do both protein and nucleotide. Also make sure that the taxon names in the sequence files adher to these rules: “illegal characters in taxon-names are: tabulators, carriage returns, spaces, ”:”, ”,”, ”)”, ”(”, ”;”, ”]”, ”[””.

```
> data(PROTEIN.fasta)
> write.fasta(sequences = PROTEIN.fasta, names = names(PROTEIN.fasta), file.out = "PROTEIN.fasta")
> UploadFile(local.file.name="PROTEIN.fasta", filetype="FASTA-0")
```

## 9.2 Muscle

```
Muscle(file.name, file.path="", job.name=NULL, args=NULL, version="Muscle-3.8.32u4",
print.curl=FALSE, aln.filetype="PHYLIP_INT", shared.username=NULL,
suppress.Warnings=FALSE)
```

`MUSCLE` is a program for creating multiple alignments of amino acid or nucleotide sequences. A range of options is provided that give you the choice of optimizing accuracy, speed, or some compromise between the two. The manual is also available here: `http://www.drive5.com/muscle/muscle_userguide3.8.html`

```
> myJobMuDP <- Muscle("DNA.fasta", aln.filetype="PHYLIP_INT", job.name="muscleDNAphyINT")

Job submitted. You can check your job using CheckJobStatus(25918)
Result file: phylip_interleaved.aln

> myJobMuDF <- Muscle("DNA.fasta", aln.filetype="FASTA", job.name="muscleDNAfasta")

Job submitted. You can check your job using CheckJobStatus(25919)
Result file: fasta.aln

> myJobMuDPP <- Muscle("DNA.fasta", aln.filetype="PHYLIP_PARS", job.name="muscleDNAphyPARS")

Job submitted. You can check your job using CheckJobStatus(25920)
Result file: phylip_pars.aln

> myJobMuDPS <- Muscle("DNA.fasta", aln.filetype="PHYLIP_SEQ", job.name="muscleDNAphySEQ")

Job submitted. You can check your job using CheckJobStatus(25921)
Result file: phylip_sequential.aln

> myJobMuDC <- Muscle("DNA.fasta", aln.filetype="CLUSTALW", job.name="muscleDNAclustalw")

Job submitted. You can check your job using CheckJobStatus(25922)
Result file: clustalw.aln

> myJobMuDM <- Muscle("DNA.fasta", aln.filetype="MSF", job.name="muscleDNAmsf")

Job submitted. You can check your job using CheckJobStatus(25923)
Result file: msf.aln

> myJobMuPP <- Muscle("PROTEIN.fasta", aln.filetype="PHYLIP_INT", job.name="musclePROTEINphyINT")

Job submitted. You can check your job using CheckJobStatus(25924)
Result file: phylip_interleaved.aln

> myJobMuPF <- Muscle("PROTEIN.fasta", aln.filetype="FASTA", job.name="musclePROTEINfasta")

Job submitted. You can check your job using CheckJobStatus(25925)
Result file: fasta.aln

> myJobMuPPP <- Muscle("PROTEIN.fasta", aln.filetype="PHYLIP_PARS", job.name="musclePROTEINphyPARS")

Job submitted. You can check your job using CheckJobStatus(25926)
Result file: phylip_pars.aln

> myJobMuPPS <- Muscle("PROTEIN.fasta", aln.filetype="PHYLIP_SEQ", job.name="musclePROTEINphySEQ")

Job submitted. You can check your job using CheckJobStatus(25927)
Result file: phylip_sequential.aln

> myJobMuPC <- Muscle("PROTEIN.fasta", aln.filetype="CLUSTALW", job.name="musclePROTEINclustalw")
```

```
Job submitted. You can check your job using CheckJobStatus(25928)
Result file: clustalw.aln
```

```
> myJobMuPM <- Muscle("PROTEIN.fasta", aln.filetype="MSF", job.name="muscleDNAmsf")
```

```
Job submitted. You can check your job using CheckJobStatus(25929)
Result file: msf.aln
```

MUSCLE outputs six alignments: `fasta.aln` (`http://en.wikipedia.org/wiki/FASTA_format`), `phylip_sequential.aln`, `phylip_interleaved.aln`, `phylip_pars.aln` (`http://www.bioperl.org/wiki/PHYLIP_multiple_alignment_format`), `clustalw.aln` (`http://meme.nbcr.net/meme/doc/clustalw-format.html`) and `msf.aln` (`http://en.wikipedia.org/wiki/MSF`).

## 9.3 Mafft

`Mafft(file.name, file.path="", type="DNA", print.curl=FALSE, version="mafftDispatcher-1.0.13100u1", args=NULL, job.name=NULL, aln.filetype="FASTA", shared.username=NULL, suppress.Warnings=FALSE)`

MAFFT is a multiple sequence alignment program for unix-like operating systems. It offers a range of multiple alignment methods, L-INS-i (accurate; for alignment of about 200 sequences), FFT-NS-2 (fast; for alignment of about 10,000 sequences), etc. See `http://mafft.cbrc.jp/alignment/software/`. The manual is also available here: `http://mafft.cbrc.jp/alignment/software/manual/manual.html`.

```
> myJobMaDF <- Mafft("DNA.fasta", job.name="mafftDNAfasta")
```

```
Job submitted. You can check your job using CheckJobStatus(25930)
Result file: mafft.fa
```

```
> myJobMaDC <- Mafft("DNA.fasta", aln.filetype="CLUSTALW", job.name="mafftDNAclustalw")
```

```
Job submitted. You can check your job using CheckJobStatus(25931)
Result file: mafft.fa
```

```
> myJobMaPF <- Mafft("PROTEIN.fasta", type="PROTEIN", job.name="mafftPROTEINfasta")
```

```
Job submitted. You can check your job using CheckJobStatus(25932)
Result file: mafft.fa
```

```
> myJobMaPC <- Mafft("PROTEIN.fasta", type="PROTEIN", aln.filetype="CLUSTALW",
+                    job.name="mafftPROTEINclustalw")
```

```
Job submitted. You can check your job using CheckJobStatus(25933)
Result file: mafft.fa
```

MAFFT outputs two alignments (both named: `mafft.fa`): FASTA (`http://en.wikipedia.org/wiki/FASTA_format`) and CLUSTALW (`http://meme.nbcr.net/meme/doc/clustalw-format.html`).

## 9.4 ClustalW

`ClustalW(file.name, file.path="", type="DNA", job.name=NULL, version="ClustalW2-2.1u1", print.curl=FALSE, args=NULL, aln.filetype="PHYLIP", shared.username=NULL, suppress.Warnings=FALSE)`

An approach for performing multiple alignments of large numbers of amino acid or nucleotide sequences is described. The method is based on first deriving a phylogenetic tree from a matrix of all pairwise sequence similarity scores, obtained using a fast pairwise alignment algorithm. See details on `http://www.clustal.org/clustal2/`.

```
> myJobCWDP <- ClustalW("DNA.fasta", job.name="clustalwDNAphylip")

Job submitted. You can check your job using CheckJobStatus(25934)
Result file: clustalw2.fa

> myJobCWDC <- ClustalW("DNA.fasta", aln.filetype="CLUSTALW", job.name="clustalwDNAclustalw")

Job submitted. You can check your job using CheckJobStatus(25935)
Result file: clustalw2.fa

> myJobCWDN <- ClustalW("DNA.fasta", aln.filetype="NEXUS", job.name="clustalwDNAnexus")

Job submitted. You can check your job using CheckJobStatus(25936)
Result file: clustalw2.fa

> myJobCWDGCG <- ClustalW("DNA.fasta", aln.filetype="GCG", job.name="clustalwDNAgcg")

Job submitted. You can check your job using CheckJobStatus(25937)
Result file: clustalw2.fa

> myJobCWDGDE <- ClustalW("DNA.fasta", aln.filetype="GDE", job.name="clustalwDNAgde")

Job submitted. You can check your job using CheckJobStatus(25938)
Result file: clustalw2.fa

> myJobCWDPIR <- ClustalW("DNA.fasta", aln.filetype="PIR", job.name="clustalwDNApir")

Job submitted. You can check your job using CheckJobStatus(25939)
Result file: clustalw2.fa

> myJobCWPP <- ClustalW("PROTEIN.fasta", type="PROTEIN", job.name="clustalwPROTEINphylip")

Job submitted. You can check your job using CheckJobStatus(25940)
Result file: clustalw2.fa

> myJobCWPC <- ClustalW("PROTEIN.fasta", type="PROTEIN", aln.filetype="CLUSTALW",
+                       job.name="clustalwPROTEINclustalw")

Job submitted. You can check your job using CheckJobStatus(25941)
Result file: clustalw2.fa

> myJobCWPN <- ClustalW("PROTEIN.fasta", type="PROTEIN", aln.filetype="NEXUS",
+                       job.name="clustalwPROTEINnexus")

Job submitted. You can check your job using CheckJobStatus(25942)
Result file: clustalw2.fa

> myJobCWPGCG <- ClustalW("PROTEIN.fasta", type="PROTEIN", aln.filetype="GCG",
+                       job.name="clustalwPROTEINgcg")

Job submitted. You can check your job using CheckJobStatus(25943)
Result file: clustalw2.fa

> myJobCWPGDE <- ClustalW("PROTEIN.fasta", type="PROTEIN", aln.filetype="GDE",
+                       job.name="clustalwPROTEINgde")

Job submitted. You can check your job using CheckJobStatus(25944)
Result file: clustalw2.fa

> myJobCWPPIR <- ClustalW("PROTEIN.fasta", type="PROTEIN", aln.filetype="PIR",
+                       job.name="clustalwPROTEINpir")
```

```
Job submitted. You can check your job using CheckJobStatus(25945)
Result file: clustalw2.fa
```

ClustalW outputs six alignments (all named: clustalw.fa): CLUSTALW http://meme.nbcr.net/meme/doc/clustalw-format.html, PHYLIP_INT http://www.bioperl.org/wiki/PHYLIP_multiple_alignment_format, NEXUS http://en.wikipedia.org/wiki/Nexus_file, GCG http://www.genomatix.de/online_help/help/sequence_formats.html#GCG, GDE http://www.cse.unsw.edu.au/~binftools/birch/GDE/overview/GDE.file_formats.html, and PIR http://www.bioinformatics.nl/tools/crab_pir.html.

## 9.5 FastTree

`Fasttree <- function(file.name, file.path="", job.name=NULL, args=NULL, type="DNA", model=NULL, gamma=FALSE, stat=FALSE, print.curl=FALSE, version="fasttreeDispatcher-1.0.0u1", shared.username=NULL, suppress.Warnings=FALSE)`

FastTree infers approximately-maximum-likelihood phylogenetic trees from alignments of nucleotide or protein sequences. See http://meta.microbesonline.org/fasttree/

```
> myJobFaDMuP <- Fasttree("phylip_interleaved.aln", file.path=paste("analyses/",myJobMuDP[[2]],
+                          sep=""), job.name="fasttreeMUSCLEdnaPHY")

Job submitted. You can check your job using CheckJobStatus(25946)

> myJobFaDCWP <- Fasttree("clustalw2.fa", file.path=paste("analyses/",myJobCWDP[[2]], sep=""),
+                          job.name="fasttreeCLUSTALWdnaPHY")

Job submitted. You can check your job using CheckJobStatus(25947)

> myJobFaDMuF <- Fasttree("fasta.aln", file.path=paste("analyses/",myJobMuDF[[2]], sep=""),
+                          job.name="fasttreeMUSCLEdnaFASTA")

Job submitted. You can check your job using CheckJobStatus(25948)

> myJobFaDCWF <- Fasttree("mafft.fa", file.path=paste("analyses/",myJobMaDF[[2]], sep=""),
+                          job.name="fasttreeMAFFTdnaFASTA")

Job submitted. You can check your job using CheckJobStatus(25949)

> myJobFaPMuP <- Fasttree("phylip_interleaved.aln", file.path=paste("analyses/",myJobMuPP[[2]],
+                         sep=""), type="PROTEIN",
+                         job.name="fasttreeMUSCLEproteinPHY")

Job submitted. You can check your job using CheckJobStatus(25950)

> myJobFaPCWP <- Fasttree("clustalw2.fa", type="PROTEIN", file.path=paste("analyses/",myJobCWPP[[2]],
+                          sep=""), job.name="fasttreeCLUSTALWproteinPHY")

Job submitted. You can check your job using CheckJobStatus(25951)

> myJobFaPMuF <- Fasttree("fasta.aln", file.path=paste("analyses/",myJobMuPF[[2]], sep=""),
+                          type="PROTEIN", job.name="fasttreeMUSCLEproteinFASTA")

Job submitted. You can check your job using CheckJobStatus(25952)

> myJobFaPCWF <- Fasttree("mafft.fa", file.path=paste("analyses/",myJobMaPF[[2]], sep=""),
+                          type="PROTEIN", job.name="fasttreeMAFFTproteinFASTA")

Job submitted. You can check your job using CheckJobStatus(25953)
```

Fasttree outputs trees in Newick format http://en.wikipedia.org/wiki/Newick_format. The placement of the root is not biologically meaningful. The local support values are given as names for the internal nodes, and range from 0 to 1, not from 0 to 100 or 0 to 1,000. If all sequences are unique, then the tree will be fully resolved (the root will have three children and other internal nodes will have two children). If there are multiple sequences that are identical to each other, then there will be a multifurcation. Also, there are no support values for the parent nodes of redundant sequences.

## 9.6 RAxML (Randomized Accelerated Maximum Likelihood)

```
RAxML(file.name, file.path="", job.name=NULL, type="DNA", model=NULL, bootstrap=NULL,
algorithm="d", multipleModelFileName=NULL, args=NULL, numcat=25, nprocs=12,
version="raxml-lonestar-7.2.8u1", print.curl=FALSE, shared.username=NULL,
substitution_matrix=NULL, empirical.frequencies=FALSE, suppress.Warnings=FALSE)
```

RAxML is a program for sequential and parallel Maximum Likelihood based inference of large phylogenetic tress. It has originall been derived from from fastDNAml which in turn was derived from Joe Felsentein's dnaml which is part of the PHYLIP package. See http://sco.h-its.org/exelixis/oldPage/RAxML-Manual.7.0.4.pdf for details.

```
> myJobRDMuP <- RAxML("phylip_interleaved.aln", file.path=paste("analyses/",myJobMuDP[[2]],
+                      sep=""), job.name="raxmlMUSCLEdnaPHY")

Job submitted. You can check your job using CheckJobStatus(25954)

> myJobRDCWP <- RAxML("clustalw2.fa", file.path=paste("analyses/",myJobCWDP[[2]], sep=""),
+                      job.name="raxmlCLUSTALWdnaPHY")

Job submitted. You can check your job using CheckJobStatus(25955)

> myJobRPMuP <- RAxML("phylip_interleaved.aln", file.path=paste("analyses/",myJobMuPP[[2]],
+                      sep=""), type="PROTEIN", job.name="raxmlMUSCLEproteinPHY")

Job submitted. You can check your job using CheckJobStatus(25956)

> myJobRPCWP <- RAxML("clustalw2.fa", file.path=paste("analyses/",myJobCWPP[[2]], sep=""),
+                      type="PROTEIN", job.name="raxmlCLUSTALWproteinPHY")

Job submitted. You can check your job using CheckJobStatus(25957)
```

For this application there are numerous output files. See pg 16-17 of the manual for complete details. RAxML outputs trees in Newick format http://en.wikipedia.org/wiki/Newick_format.

## 9.7 PHYLIP-Parsimony 3.69

```
PHYLIP_Pars(file.name, file.path="", job.name=NULL, type="DNA", print.curl=FALSE,
shared.username=NULL, suppress.Warnings=FALSE)
```

PHYLIP is a free package of programs for inferring phylogenies. It is distributed as source code, documentation files, and a number of different types of executables. The web page: http://evolution.genetics.washington.edu/phylip/doc/main.html, by Joe Felsenstein of the Department of Genome Sciences and the Department of Biology at the University of

Washington, contain information on PHYLIP. PHYLIP (the PHYLogeny Inference Package) is a package of programs for inferring phylogenies (evolutionary trees). Methods that are available in the package include parsimony, distance matrix, and likelihood methods, including bootstrapping and consensus trees.

```
> myJobPDMuPP <- PHYLIP_Pars("phylip_pars.aln", file.path=paste("analyses/",myJobMuDPP[[2]],
+                             sep=""), job.name="phylipMUSCLEdnaPHYpars")
```

```
Job submitted. You can check your job using CheckJobStatus(25958)
```

```
> myJobPPMuPP <- PHYLIP_Pars("phylip_pars.aln", file.path=paste("analyses/",myJobMuPPP[[2]],
+                             sep=""), type="PROTEIN", job.name="phylipMUSCLEproteinPHYpars")
```

```
Job submitted. You can check your job using CheckJobStatus(25959)
```

`PHYLIP Parsimony` outputs trees in Newick format `http://en.wikipedia.org/wiki/Newick_format`.

## 9.8 Genome Wide Association Study models

```
> UploadFile("simulation1.map")
> UploadFile("simulation1.ped")
> UploadFile("geno_test.tfam")
> UploadFile("geno_test.tped")
```

## 9.9 PLINK

`PLINK(file.list="", file.path="", job.name=NULL, association.method="-assoc", no.sex=TRUE, args=NULL, print.curl=FALSE, multi.adjust=TRUE, version="plink-1.07u1", shared.username=NULL, suppress.Warnings=FALSE)`

`PLINK` is an open-source whole genome association analysis toolset, designed to perform a range of basic, large-scale analyses in a computationally efficient manner, check `http://pngu.mgh.harvard.edu/~purcell/plink/` for details.

```
> myJobPLINKT <- PLINK(file.list=list("geno_test.tfam","geno_test.tped"), job.name="PLINKT")
```

```
Job submitted. You can check your job using CheckJobStatus(25960)
```

```
> myJobPLINKR <- PLINK(file.list=list("simulation1.map","simulation1.ped"), job.name="PLINKR")
```

```
Job submitted. You can check your job using CheckJobStatus(25961)
```

There are many output files possible, `http://pngu.mgh.harvard.edu/~purcell/plink/reference.shtml#output`

## 9.10 PLINK Conversion

`PLINKConversion(file.list="", file.path="", output.type="-recode", job.name=NULL, shared.username=NULL, print.curl=FALSE, version="plink-1.07u1", suppress.Warnings=FALSE)`

This function converts the standard PLINK file formats (Regular (ped/map), Transposed (tped/tfam), and Binary (bed/bim/fam)) to various other PLINK file formats.

```
> myJobPLINKCT <- PLINKConversion(file.list=list("geno_test.tfam","geno_test.tped"), job.name="PCT",
>                                 out.basename="plinkout")
```

```
Job submitted. You can check your job using CheckJobStatus(25962)
```

```
> myJobPLINKCR <- PLINKConversion(file.list=list("simulation1.map","simulation1.ped"),
+                                 output.type="--recode --transpose", job.name="PCR")
```

```
Job submitted. You can check your job using CheckJobStatus(25963)
```

There are many output files possible, `http://pngu.mgh.harvard.edu/~purcell/plink/reference.shtml#output`

## 9.11 FaST-LMM (Factored Spectrally Transformed Linear Mixed Models)

`FaST_LMM(input.file.list="", ALL.file.path="", print.curl=FALSE, sim.file.list=NULL, pheno.file.name=NULL, mpheno=1, args=NULL, covar.file.name=NULL, job.name=NULL, version="FaST-LMM-1.09u1", shared.username=NULL, suppress.Warnings=FALSE)`

`FaST-LMM (Factored Spectrally Transformed Linear Mixed Models)` is a program for performing genome-wide association studies (GWAS) on large data sets. FaST-LMM is described more fully at `http://www.nature.com/nmeth/journal/v8/n10/abs/nmeth.1681.html`, and also at `http://fastlmm.codeplex.com/`

```
> myJobFaST_LMMT <- FaST_LMM(input.file.list=list("geno_test.tfam","geno_test.tped"),
>                            job.name="FaST_LMMT")
```

```
Job submitted. You can check your job using CheckJobStatus(25964)
```

```
> myJobFaST_LMMR <- FaST_LMM(input.file.list=list("simulation1.map","simulation1.ped"),
>                            job.name="FaST_LMMR")
```

```
Job submitted. You can check your job using CheckJobStatus(25965)
```

Not all information on the FaST-LMM model is here, see the FaST-LMM website `http://fastlmm.codeplex.com/`, or the FaST-LMM manual for more information.

# 10 Creating workflows

Finally, each of these steps can be combined to generate multi-step analyses. This has the benefit of reducing errors that can occur when manually running each application and, more importantly, ensures that results are reproducible. In the following example, a user starts with unaligned sequences on her or his local computer and ends with aligned sequences and a phylogenetic tree with all applications running on the iPlant servers.

## 10.1 Workflow One

This first workflow takes an amino acid fasta file, uses `MUSCLE` to get a `PHYLIP_PARS` alignment type. A couple things about this alignment; it is only available from `MUSCLE` and this alignment is very specific to the `PHYLIP 3.69` model. The `PHYLIP` model then produces a tree. Note: this is the only way to do this workflow.

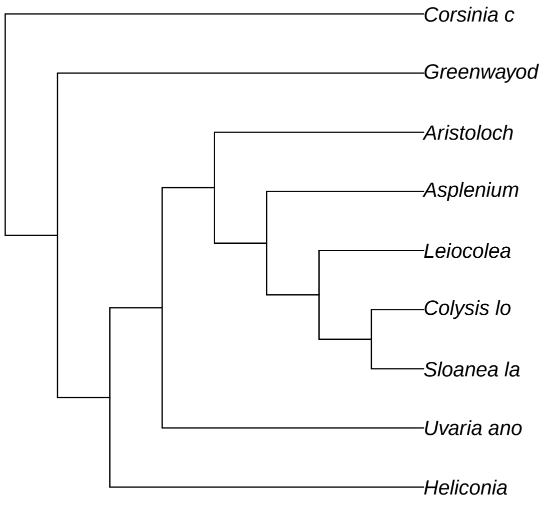
```
> myJobW1MP <- Muscle("PROTEIN.fasta", aln.filetype="PHYLIP_PARS", job.name="muscleWORKFLOW1protein")
```

```
Job submitted. You can check your job using CheckJobStatus(25966)
Result file: phylip_pars.aln
```

```
> Wait(myJobW1MP[[1]], minWait, maxWait)
> myJobW1PPP <- PHYLIP_Pars("phylip_pars.aln", file.path=paste("analyses/",myJobW1MP[[2]],
>                            sep=""), type="PROTEIN", job.name="phylipWORKFLOW1protein")
```

```
Job submitted. You can check your job using CheckJobStatus(25967)
```

```
> Wait(myJobW1PPP[[1]], minWait, maxWait)
> RetrieveJob(myJobW1PPP[[1]], c("outtree.nwk"))
> read.tree(paste(getwd(), myJobW1PPP[[2]], "outtree.nwk", sep="/")) -> Tree
```



## 10.2 Workflow Two

The second workflow takes the same amino acid fasta file and this time it uses `Mafft` to get a `FASTA` alignment type. `MUSCLE` also can output a fasta alignment. The `FastTree` model is then used to make the tree.

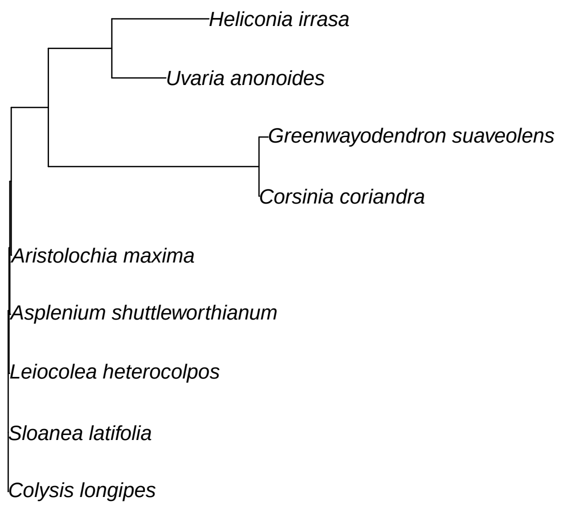
```
> myJobW2CWD <- Mafft("PROTEIN.fasta", type="PROTEIN", job.name="mafftPROTEINfasta")
```

```
Job submitted. You can check your job using CheckJobStatus(25968)
Result file: mafft.fa
```

```
> Wait(myJobW2CWD[[1]], minWait, maxWait)
> myJobW2FaD <- Fasttree("mafft.fa", type="PROTEIN", file.path=paste("analyses/",myJobW2CWD[[2]],
+                         sep=""), job.name="fasttreeCLUSTALWfasta")
```

```
Job submitted. You can check your job using CheckJobStatus(25969)
```

```
> Wait(myJobW2FaD[[1]], minWait, maxWait)
> RetrieveJob(myJobW2FaD[[1]], c("fasttree.nwk"))
> read.tree(paste(getwd(), myJobW2FaD[[2]], "fasttree.nwk", sep="/")) -> Tree
```



## 10.3 Workflow Three

The third workflow is again dealing with `FastTree`. `FastTree` can take either a FASTA alignment or a phylip interleaved alignment as inputs. Now `ClustalW` takes a nucleotide fasta file to get a `PHYLIP INTERLEAVED` alignment type. `MUSCLE` also can output that alignment. The `FastTree` model is then used to make the tree.

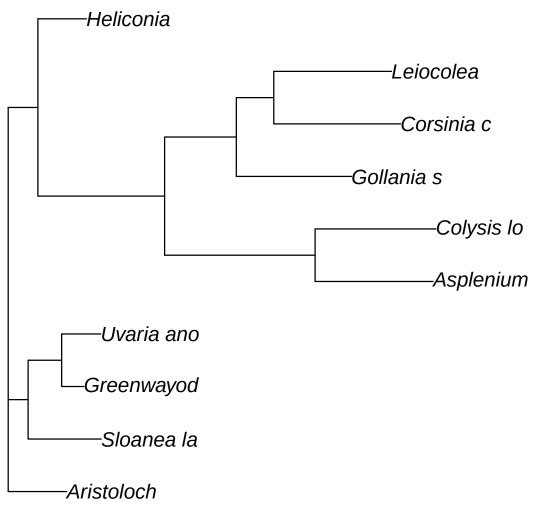
```
> myJobW3MuP <- ClustalW("DNA.fasta", job.name="clustalwDNAfasta")

Job submitted. You can check your job using CheckJobStatus(25970)
Result file: clustalw2.fa

> Wait(myJobW3MuP[[1]], minWait, maxWait)
> myJobW3FaP <- Fasttree("clustalw2.fa", file.path=paste("analyses/",myJobW3MuP[[2]], sep=""),
+                         job.name="fasttreeMUSCLEdna")

Job submitted. You can check your job using CheckJobStatus(25971)

> Wait(myJobW3FaP[[1]], minWait, maxWait)
> RetrieveJob(myJobW3FaP[[1]], c("fasttree.nwk"))
> read.tree(paste(getwd(), myJobW3FaP[[2]], "fasttree.nwk", sep="/")) -> Tree
```



## 10.4 Workflow Four

The fourth workflow is using `RAxML`. `MUSCLE` takes a nucleotide fasta file to get a `PHYLIP INTERLEAVED` alignment type. `ClustalW` also can output that alignment. The `RAxML` model is then used to make the tree.

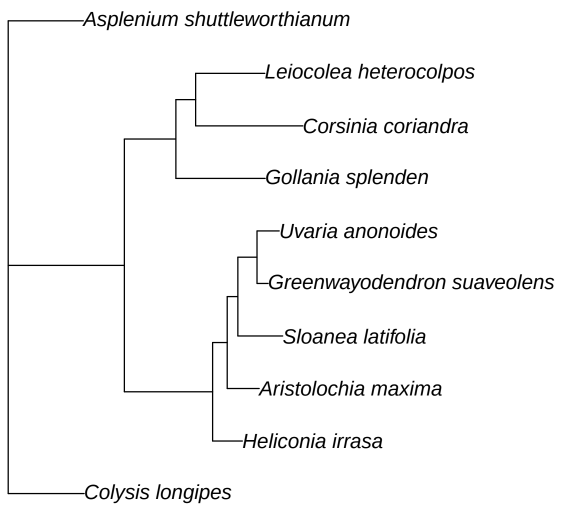
```
> myJobW4MuP <- Muscle("DNA.fasta", aln.filetype="PHYLIP_INT", job.name="muscleWORKFLOW4dna")

Job submitted. You can check your job using CheckJobStatus(25972)
Result file: phylip_interleaved.aln

> Wait(myJobW4MuP[[1]], minWait, maxWait)
> myJobW4RP <- RAxML("phylip_interleaved.aln", file.path=paste("analyses/",myJobW4MuP[[2]], sep=""),
+                     job.name="raxmlWORKFLOW4dna")

Job submitted. You can check your job using CheckJobStatus(25973)

> Wait(myJobW4RP[[1]], minWait, maxWait)
> RetrieveJob(myJobW4RP[[1]], c("RAxML_bestTree.nwk"))
> read.tree(paste(getwd(), myJobW4RP[[2]], "RAxML_bestTree.nwk", sep="/")) -> Tree
```


Asplenium shuttleworthianum
Leiocolea heterocolpos
Corsinia coriandra
Gollania splenden
Uvaria anonoides
Greenwayodendron suaveolens
Sloanea latifolia
Aristolochia maxima
Heliconia irrasa
Colysis longipes