

### MY FTP INPUT FILES FROM CUT DATA FOR THE DESHUTES

### This script creates input files for the MyFTP program from text output from FVS and from raster grids of planning units

### Please contact jkreitler@usgs.gov with any questions

- # 1. Calculates stand proportions per planning unit
- # 2. Converts cutlist text to a data table and outputs master list ~5000 records each
- # 3. Peices together previous lists into a single cutData data table
- # 4. Creates MyFTP input
- # 5. Condences MyFTP input

#set the directory to the folder that contains the cutdata

forestdirectory <- "C:\\Users\\cschloss\\Documents\\forest\\"

#set the directory to the folder that contains the spatial layers

spatialdirectory <- "C:\\Users\\cschloss\\Documents\\forest\\GNN\\"

#name of the cutlist file - changed from .trl to .txt

cutListFile <- "ThinBTA100\_ALL.txt"

#####

#####

#### 1. Stand Proportions Per Planning Unit

#### Calculates the proportion of each stand whose total area is greater than >100ha in each planning unit

#### Stands with area greater than 100ha is determined from a provided DBF file list rather than the raster to be consistant with other filters for the same project

#### A. Reads in 2 raster files - a planning unit raster file and a raster file with stand ID's - these must have same projection, cellsize, and extent

#### B. Determines # of cells of each stand type in each planning unit and converts to proportions and stores in master list

```
library(rgdal) #library to read in rasters
```

```
library(foreign) #library to read in dbf files
```

```
puStandProp<-list() # blank list to fill with proportional data per planning unit
```

```
#read planning unit raster
```

```
puRast<-readGDAL(paste(spatialdirectory, "/desPU", sep=""))
```

```
#create an index of cells with planning units (i.e. not NoData)
```

```
puRastIndex<-which(is.na(puRast@data$band1)==F)
```

```
#remove Nodata cells
```

```
puRast<-puRast@data$band1[puRastIndex]
```

```
#create list of planning unit names
```

```
allPuNames<-unique(puRast)
```

```
#read in raster of stand id information
```

```
standRast<-readGDAL(paste(spatialdirectory, "/gnndesownmaj", sep=""))
```

```
#limit to just areas within planning units
```

```
standRast<-standRast@data$band1[puRastIndex]
```

```
#read in dbf file list of just the stand ids to incorporate (i.e. stand >100ha)
```

```
standLarge<-read.dbf(paste(spatialdirectory, "GNNDesOwnFCID100ac.dbf", sep=""))
```

```
standLarge<-standLarge$VALUE
```

```
#loop through every planning unit
```

```
for(pu in allPuNames){
```

```

print(pu)

#create an index of which cells are in the specific planning unit 'pu'
puIndex<-which(puRast==pu)

#pull out the stand info for JUST the stands that are in the specific planning unit 'pu'
standInPu<-standRast[puIndex]

#remove NoData cells from stand layer
standInPu<-standInPu[which(is.na(standInPu)==F)]

# make sure there is at least ONE stand in the planning unit
if(length(standInPu)>0){

#tabulate the number of cells of each stand type in the planning unit
tablePU<-as.data.frame(table(standInPu))

#remove all rows that are not in the LARGE STANDS dbf file (read in before the loop)
tablePU<-tablePU[which(tablePU$standInPu%in%standLarge==T),]

#calculate proportion of each stand in the pu from cell frequency values
tablePU$proportion<-tablePU$Freq/sum(tablePU$Freq)

#add proportion to master LIST of proportions for each planning unit and name it as puX
puStandProp[[paste("pu", pu, sep="")]<-tablePU

}# end if statement making sure that there is at least 1 stand
} #end loop through planning units

```

```
#####
```

```
#####
```

```
#### 2. Converting cutlist output to a single file with a stand ID column and no headers
```

```
#### Before running this script divide into several files and run each separately or this will take a  
LONG LONG time to run.
```

```
#### Ultimately strips headers and adds stand id to data to make a master list of data for all stands
```

```
#### Because this is done in stages to avoid both "skipping" hundreds of thousands of lines and  
"rbinding" to hundreds of thousands of lines - which is time consuming - this must be peiced back  
together before continuing on with the next script
```

```
####!!!! see note (below)
```

```
# a list of row names (because the names are on seperate lines they cannot be read in together - if  
output row names or type changes this needs to be manually updated!!
```

```
cutRowName<-c("STAND", "TREE_NUMBER", "TREE_INDX", "SP_CD", "SP_NO", "TR_CL", "SS_CD",  
"PNT_NUM", "TREES_PER_ACRE", "MORTAL_PER_ACRE", "CURR_DIAM", "DIAM_INCR", "CURR_HT",  
"HT_INCR", "CR", "MAX_CW", "MS", "BA_%-TILE", "POINT_BAL", "TOT_CU_F_VOL", "MCH_CU_FT_VOL",  
"MCH_BD_FT_VOL", "MC_DF", "BF_DF", "TRC_HT")
```

```
# set up a blank file for the cut data (fvs output?)
```

```
cutData<-as.data.frame(matrix(rep(NA, length(cutRowName)), ncol=length(cutRowName), nrow=1))
```

```
#add names to the data frame for the cut data
```

```
names(cutData)<-cutRowName
```

```
#line length is collected as a way of knowing when the loop should finish - when 2 blank lines are read  
in sequentially there is no more data to collect
```

```
lengthLine<-1
```

```
lengthLine2<-1
```

```
#the number of lines to skip when reading in the data - start as 0 to read in the first line
```

```
skipLine <- 0
```

```
writeNow<-5000
```

#the following loop reads in all of the lines of data - if it is a header line it looks for the stand ID and moves on to read in the data and add the stand ID

#when it encounters another header, it looks for and adds the proper stand ID before proceeding.

#start loop through all of the lines in the data - as long as there are not 2 consecutive blank lines (end)

```
while(lengthLine>0|lengthLine2>0){
```

```
  lengthLine2<-lengthLine
```

```
  #read in a line of data from the cut list
```

```
  cutLine <- scan(paste(forestdirectory, cutListFile, sep=""), what="raw", nlines=1, skip=skipLine)
```

```
  #find length of the data read in
```

```
  lengthLine <- length(cutLine)
```

```
  #proceed if the data has at least 4 elements and it is the first line of the data header output (aka start to new data section)
```

```
  if(lengthLine>3&&(cutLine[2]=="FOREST"& cutLine[3]=="VEGETATION" & cutLine[4]=="SIMULATOR")){
```

```
    #collect stand information for the following data
```

```
    stand<-scan(paste(forestdirectory,cutListFile, sep=""), what="raw", nlines=1, skip=skipLine+2)[6]
```

```
    print(stand)
```

```
    #advance skip so that we skip the rest of the header (blank lines, row names, other info)
```

```
    skipLine=skipLine+6
```

```
    #read in the first line of actual data (no header)
```

```
    cutLine <- scan(paste(forestdirectory, cutListFile, sep=""), what="raw", nlines=1, skip=skipLine)
```

```
  #loop through data for this stand and page - stop adding to dataframe when a new header is encountered
```

```
  while(lengthLine>3&&(cutLine[2]!="FOREST"& cutLine[3]!="VEGETATION" & cutLine[4]!="SIMULATOR")){
```

```
    #add row of data to master list of cut data
```

```

cutData<-rbind(cutData, c(stand, cutLine))

#advance to next line
skipLine=skipLine+1

#read in next line of data
cutLine <- scan(paste(forestdirectory, cutListFile, sep=""), what="raw", nlines=1, skip=skipLine)
} # end while line that was just read in is still data and not the next header

} # end if header is reached (start of new data)

if(skipLine>writeNow){
#remove the NA line from the master list of data
cutData<-cutData[which(is.na(cutData$STAND)==F),]

#write a csv of first 5000 lines of data
write.csv(cutData, paste(forestdirectory,"//master_lists//master_cut_", writeNow, ".csv", sep=""))

#advance writeNow so that it doesnt write out data for another 5000 lines
writeNow=writeNow+5000

#setup new blank cutData to keep speed going
cutData<-as.data.frame(matrix(rep(NA, length(cutRowName)), ncol=length(cutRowName), nrow=1))

#add names to the data frame for the cut data
names(cutData)<-cutRowName

}# end if write data out

} # end while not 2 blank lines - aka while data continues

#Write out the last peice - because if statement isnt triggered

#remove the NA line from the master list of data

```

```
cutData<-cutData[which(is.na(cutData$STAND)==F),]

#write a csv of last part of data

write.csv(cutData, paste(forestdirectory,"//master_lists//master_cut_", writeNow, ".csv", sep=""))
```

```
#####
```

```
#####
```

```
#### 3. Piecing data back together - because split up to reduce time
```

```
#### A. reads in each tabulated cutlist csv
```

```
#### B. pieces into a single file
```

```
parts<-seq(5000, writeNow, by=5000)
```

```
for(p in parts){

  filename<-paste(forestdirectory, "master_cut_", p, ".csv", sep="")

  partX<-read.csv(filename)

  partX<-partX[,-which(names(partX)=="X")]

  print(p)

  print(paste(head(partX$STAND)[1], tail(partX$STAND)[1]))

  #print(range(partX$STAND))

  if(p==5000){whole<-partX}

  if(p>5000){whole<-rbind(whole, partX)}

}
```

```
cutData<-whole
```

```
#####
```

```
#####
```

#### #### 4. Creating Input for MyFTP program

#### A. The following code mostly just pulls the proper data in the proper order for each planning unit

#### B. Also it adjusts the tree per acre column by the proportion of the planning unit made up by that stand type, uses stand for Tree ID and planning unit ID for Stand ID

#### C. An output excel file will be created for each planning unit.

```
#cutData<-read.csv()

#setup a blank output file format with proper columns and column names

blank<-data.frame(StandId=NA, StandYear=NA, TreeId=NA, Species=NA, Diameter=NA, Height=NA,
TPA=NA)
```

```
#start loop through planning units

for (pu in allPuNames){

  puDataNew<-blank

  #pull out proportional planning unit stand data from list created in step 1 (above)

  puData<- puStandProp[[paste("pu", pu, sep="")]

  #loop through each stand that exists in the planning unit

  for(stand in puData$stand){

    #limit cut data master file (created in step 2 above) to just the cut data for that stand

    cutStand<-cutData[which(cutData$STAND==stand),]

    if(length(cutStand[,1])>0){

      #pull in proper data (switch stand ID to planning unit id, swithc tree id to stand id, convert trees per
      acre based on stand proportion)
```

```

newData<-data.frame(StandId=rep(pu, length(cutStand[,1])),StandYear=rep(2008,
length(cutStand[,1])), Treeld=rep(stand, length(cutStand[,1])), Species=cutStand$SP_CD,
Diameter=cutStand$CURR_DIAM, Height=cutStand$CURR_HT,
TPA=as.numeric(cutStand$TREES_PER_ACRE)*puData$proportion[which(puData$stand==stand)])

#add the data rows for that stand to the master data for the planning unit

puDataNew<-rbind(puDataNew, newData)

}

if(length(cutStand[,1])!=0){print(paste(pu, stand))}

} # end loop through stands in the planning unit

puDataNew<-puDataNew[which(is.na(puDataNew$StandId)==F),]

#write out data for each planning unit "input file"

write.csv(puDataNew, paste(forestdirectory,"InputFiles\\inputPU", pu, ".csv", sep="" ), row.names=F)

}# end loop through planning unit

#####

#####

#### 5. Condensing the MyFTP files (created above)

### A. All rows that are in the same stand, same planning unit and are the same species and within
1inch increments (0.6-1.5, 1.6-2.5, 2.6-3.5 etc) will be merged to a single record

### B. heights will be the average of all the heights and TPA will be the sum of all the TPA

### C. An output excel file will be created for each planning unit.

#cutData<-read.csv()

#setup a blank output file format with proper columns and column names

```

```
blank<-data.frame(StandId=NA, StandYear=NA, Treeld=NA, Species=NA, Diameter=NA, Height=NA,
TPA=NA)
```

```
#this is a list of the numbers to round down (i.e ending in .5)
```

```
floors<-seq(0.5, 12.5, by=1)
```

```
for (pu in allPuNames){
```

```
  #start a blank file for each planning unit
```

```
  puFix<-blank
```

```
  #read in csv
```

```
  puDataX<-read.csv(paste(forestdirectory,"InputFiles\\inputPU", pu, ".csv", sep="" ))
```

```
  #replace values that end in .5 with the rounded down integer
```

```
  puDataX$Diameter[which(puDataX$Diameter%in%floors)]<-
floor(puDataX$Diameter[which(puDataX$Diameter%in%floors)])
```

```
  #round all other integers according to normal rounding rules
```

```
  puDataX$Diameter<-round(puDataX$Diameter, digits=0)
```

```
  # list of stands in the planning unit
```

```
  standlist<-unique(puDataX$Treeld)
```

```
  #start loop through stands
```

```
  for(st in standlist){
```

```
    #data subsetted for just that stand
```

```
    standX<-puDataX[which(puDataX$Treeld==st),]
```

```
    # list of all species in the stand in the planning unit
```

```
    specieslist<-unique(standX$Species)
```

```
    # loop through species
```

```

for(sp in as.character(specieslist)){

  # data subsetted for just that species

  speciesX<-standX[which(as.character(standX$Species)==sp),]

  #list of all the unique diameters - these diameters have already been binned to the nearest integer
  (or lowest integer if ending in .5)

  diameterlist<-unique(speciesX$Diameter)

  #loop through diameters

  for(d in diameterlist){

    #calculate new TPA value by adding all TPA's that have the same diameter, same species, same
    stand, and same planning unit

    newTPA<-sum(speciesX$TPA[which(speciesX$Diameter==d)])

    #calculate new height value by taking the mean of all the heights that have the same diameter,
    same species, same stand, and same planning unit

    newH<-round(mean(speciesX$Height[which(speciesX$Diameter==d)]), digits=2)

    # make a row with the new tpa and heights

    row<-c(pu, 2008, st, sp, d, newH, newTPA)

    puFix<-rbind(puFix, row)

  } # end loop through diameters

}# end loop through speices

} #end loop through stand

#remove na row from data

puFix<-puFix[which(is.na(puFix$StandId)==F),]

#write out input file csvs

write.csv(puFix, paste(forestdirectory,"InputFiles\\inputPU_binDiam_", pu, ".csv", sep=""),
row.names=F)

} # end loop through planning units

```

