Griffith UNIVERSITY

Ubuntu 8.10 AD Instructions

Developed by Matthew Lye, CTS and Anthony Thyessen, RCS
Prepared by Matthew Lye, CTS

SANITISED VERSION

VERSION 1.3

Griffith UNIVERSITY

1 Overview and Assumptions

This document assumes that you are adapting an image that already has the correct configuration for student networked homes as outlined in Ubuntu 8.10 Linux Lab Build Instructions version 1.1.

This document replaces section 2.3 for the purpose of configuring the Griffith University Linux student computing environment to connect to Active Directory.

2 Documentation Guide

Linux is an OS that maintains two different environments. Both have full control over the system unlike the windows DOS and GUI environment. As such many changes are easier to make via the terminal or command line environment.

For this document any command shown as below is a terminal command and should by entered into a terminal exactly as shown:

```
echo This is a terminal command.
```

Some changes are easier to make via the GUI and are shown in a 'path' format. This takes the form of a heading and arrows to point to the next heading to click until the path is found. For example:

System → Administration → Printing

This command would take you through the menu to the printing control panel.

3 Configuration Instructions

This method of implementing Active Directory based authentication does not require the computer to be connected to the domain. Active Directory is used for password authentication and the mounting of network drives only.

3.1 Configuring and Installing Kerberos

3.1.1 Installing and Configuring Kerberos

Install the Kerberos modules for PAM with the following command:

```
sudo apt-get install libpam-krb5 krb5-user krb5-config
```

Edit the Kerberos configuration file to contain our configuration

Open the Kerberos file:

```
sudo nano /etc/krb5.conf
```

Replace the contents with the following:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmin.log

[libdefaults]
 default_realm = STUDENT.GRIFFITH.EDU.AU
 dns_lookup_realm = true
 dns_lookup_kdc = true
 ticket_lifetime = 4h
 forwardable = yes

[realms]
 STUDENT.GRIFFITH.EDU.AU = {
  kdc = student.griffith.edu.au:88
  admin_server = student.griffith.edu.au:749
  default_domain = student.griffith.edu.au
 }

 STAFF.GRIFFITH.EDU.AU = {
  kdc = staff.griffith.edu.au:88
  admin_server = staff.griffith.edu.au:749
  default_domain = staff.griffith.edu.au
 }

[domain_realm]
 .student.griffith.edu.au = STUDENT.GRIFFITH.EDU.AU
 student.griffith.edu.au = STUDENT.GRIFFITH.EDU.AU
 .staff.griffith.edu.au = STAFF.GRIFFITH.EDU.AU
 staff.griffith.edu.au = STAFF.GRIFFITH.EDU.AU

[appdefaults]
 pam = {
   debug = false
   ticket_lifetime = 36000
   renew_lifetime = 36000
```

```
    forwardable = true
    krb4_convert = false
}
```

3.1.2 Testing Kerberos

As part of the installation package previously installed we have three commands that can be used to test connection to the AD environment.

kinit: check out ticket

klist: list checked out tickets

kdestroy: kill tickets

3.2 Configuring PAM.

PAM is the underlying architecture used for all linux logins. In this process we are instructing PAM to use pam_krb5.so (Kerberos/AD Authentication) instead of the previous method of using pam_ldap (NDS Authentication).

3.2.1 Configuring Login Process

Set default Kerberos config for pam (configures pam to use Kerberos quickly):

```
sudo auth-client-config -p kerberos_example -a
```

Configure each PAM component

Open **PAM Auth**:

```
sudo nano /etc/pam.d/common-auth
```

Replace the contents of the file with the following:

```
#
# /etc/pam.d/common-auth - authentication settings common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authentication modules that define
# the central authentication scheme for use on the system
# (e.g., /etc/shadow, LDAP, Kerberos, etc.).  The default is to use the
# traditional Unix authentication mechanisms.
#
# As of pam 1.0.1-5, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# must have a valid shell and group
auth     required     pam_shells.so
auth     required     pam_group.so

#allow local login, ignore broken shadow entry
auth     sufficient   pam_unix.so try_first_pass
auth     requisite    pam_succeed_if.so uid > 1000 quiet

#attempt to mount home directory
auth     required     pam_mount.so use_first_pass
```

```
#allow valid AD login
auth sufficient  pam_krb5.so use_first_pass realm=STAFF.GRIFFITH.EDU.AU
auth sufficient  pam_krb5.so use_first_pass realm=STUDENT.GRIFFITH.EDU.AU


#deny access if none of the above are valid
auth     required     pam_deny.so
```

Open **PAM Account**:

```
sudo nano /etc/pam.d/common-account
```

Replace the contents of the file with the following:

```
#
# /etc/pam.d/common-account - authorization settings common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authorization modules that define
# the central access policy for use on the system.  The default is to
# only deny service to users whose accounts are expired in /etc/shadow.
#
# As of pam 1.0.1-5, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.
#

# user must have a valid shell
account     required     pam_shells.so

# allow local account, ignore broken shadow entry
account     required     pam_unix.so broken_shadow

#UID Validation Check
account     sufficient   pam_succeed_if.so uid < 1000 quiet

#Kerberos ticket check
account     [default=bad success=ok user_unknown=ignore] pam_krb5.so

#If ticket check succeeds, allow login
account     required     pam_permit.so
```

Open **PAM Password**:

```
sudo nano /etc/pam.d/common-password
```

Remove or comment out all entries in the file. Griffith policy prevents users from being allowed to change AD passwords on the local machine.

Open **PAM Session**:

```
sudo nano /etc/pam.d/common-session
```

Replace the contents of the file with the following:

```
#
# /etc/pam.d/common-session - session-related modules common to all #services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define tasks to be performed
# at the start and end of sessions of *any* kind (both interactive and
# non-interactive).
#
# As of pam 1.0.1-5, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.


session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
session [default=ignore success=ok] pam_succeed_if.so service in crond quiet use_uid
session     required      pam_unix.so
session     sufficient    pam_succeed_if.so uid < 1000 quiet
session     sufficient    pam_succeed_if.so service in su-l quiet
session     optional      pam_mount.so
session     sufficient    pam_krb5.so realm=STUDENT.GRIFFITH.EDU.AU
session     sufficient    pam_krb5.so realm=STAFF.GRIFFITH.EDU.AU
session     required      pam_deny.so
```

3.2.2 Mounting H-Drives

The mounting of student H-Drives is done through the PAM Mount module calling a custom developed script.

This mounting script has only been tested under Samba 3, it should be re-examined once Samba 4 becomes the default standard.

Install the applications required:

```
sudo aptitude install libpam-mount libexpect-perl libio-pty-perl libio-stty-perl
samba-client samba-common smbfs
```

Copy the script from Ent to the local machine:

```
sudo cp /source/linux_extras/update/mount.ad /sbin/mount.ad
```

A full copy of the script with instructions found in Appendix A

Now edit pam mount to call our new script by opening the PAM Mount configuration script:

```
sudo nano /etc/security/pam_mount.conf.xml
```

Find the section <!-- Volume definitions --> and add this line:

```
<volume fstype="ad" path="%(USER)" mountpoint="/mnt/H_%(USER)"/>
```

Find the tag <umount>umount %(MNTPT)</umount> and replace it with:

```
<umount>/sbin/umount.%(FSTYPE) %(MNTPT)</umount>
```

link mount script to be used to unmount and set permissions:

```
sudo ln -s /sbin/mount.ad /sbin/umount.ad
sudo chmod 700 /sbin/mount.ad
sudo chmod 700 /sbin/umount.ad
```

4 Appendix A: Mount Script and Installation Instructions

Create the script on the machine:

```
sudo nano /sbin/mount.ad
```

Copy the following into the file:

```
#!/usr/bin/perl
#
#  mount.ad user mount-point [-o options]
#  mount.ad -u (user|mount-point)
#  umount.ad (user|mount-point)
#
# Read a password from stdin (generally from a "pam_mount") invocation, but
# will prompt for one from a TTY if present.  Find the users AD H-Drive,
# and mount it at the given mount-point (creating the directory). Also create
# a symbolic link from 'H' in users home to that mount (as the user).
#
# Any and all mount options are completely ignored at this time.
#
###
#
# Anthony Thyssen      24 September 2009
#
use strict;
use FindBin;
use Expect;
use Sys::Syslog;

my $PROGNAME = $FindBin::Script;
$ENV{PATH}="/usr/bin:/bin:/sbin";

sub Usage {
  print STDERR @_, "\n";
  @ARGV = ( "$FindBin::Bin/$PROGNAME" );
  while( <> ) {
    next if 1 .. 2;
    last if /^###/;
    s/^#$//; s/^# //;
    print STDERR;
  }
  exit 10;
}

# debugging level    1 = log calls and Errors only
#                    2 = Intermediate Variable Results
#                    3 = Sub-Command Calls and Interaction
my $DEBUG = 0;
my $LOG_DEBUG = 0;   # log to "/tmp/mount_ad.log"

# ------------------------------------------------------------------------
#  Hardcoded elements

my @AD_Types = qw( student staff );   # the trees in the AD forest

my %AD_Domain;    # EG:  STUDENT-TEST
map { $AD_Domain{$_} = uc($_)}  @AD_Types;

my %AD_Server;    # EG:  student.griffith.edu.au
map { $AD_Server{$_} = $_."griffith.edu.au" }  @AD_Types;
```

```
# Not needed if "rpcclient" information method used
my $OPTION_RPCCLIENT = 1;  # use rpcclient,  no need to work out a DN
my %LDAP_DN = (
     staff   => "OU=Staff",    # the realm is worked out from Server name
     student => "OU=Students", # <== Note that it is "Students" with an 's'!!!
);
map { ( $LDAP_DN{$_} .= $AD_Server{$_} ) =~ s/\./,DN=/; } @AD_Types;

# Not needed if direct DFS mount path used
# This is needed as Samba v3.2.14 has broken DFS referral handling
my $OPTION_LOCAL_DFS = 1;  # do DFS redirect locally, using the following lookup
my %AD_DFS = (   # DFS Equivelent: mapping user type and campus to real home
    'staff,na'   => "breeze-staff.staff.griffith.edu.au",
    'staff,gc'   => "sneeze-staff.staff.griffith.edu.au",
    'student,na' => "breeze-stud.student.griffith.edu.au",
    'student,gc' => "sneeze-stud.student.griffith.edu.au",
);
map { $AD_DFS{$_} = "//$AD_DFS{$_}/users\$"; }  keys %AD_DFS;

# The mount commands to use
my $LIST_mount = "/bin/mount -t cifs";
my $CIFS_mount = "/sbin/mount.cifs";
my $CIFS_umount = "/sbin/umount.cifs";

# ------------------------------------------------------------------------
# Initialisation and Sub-routines


# Debug tracing -- if no TTY log to  "/tmp/mount_ad.log"
if ( $DEBUG ) {
  if ( $LOG_DEBUG ) {
    open(STDOUT, ">>/tmp/mount_ad.log") || die;
    open(STDERR, ">&STDOUT");
    chmod(0600, "/tmp/mount_ad.log") || die;
  }
  chomp( my $date = qx(date +'%Y-%m-%d %R:%S') );
  print "$date $0 @ARGV\n";
  #system('id');
  if ( $DEBUG == 1 ) { # Turn off if only command call logging
    $DEBUG=0;   # Turn off if only command call logging
  }
}

sub ErrorExit {
  openlog($PROGNAME, 'ndelay', 'LOG_AUTHPRIV');
  syslog('LOG_ERR', '%s', "@_");
  closelog();
  exit(1);
}

sub OkExit {
  openlog($PROGNAME, 'ndelay', 'LOG_AUTHPRIV');
  syslog('LOG_INFO', '%s', "@_");
  closelog();
  exit(0);
}

sub FindMountingUser {
  # Try to find the username for the file system on this mount_point
  my $mount_point = shift;
  my $mounts = `$LIST_mount`;
  my ( $user ) = $mounts =~ /\W([a-z][a-z0-9]*) on $mount_point\s/;
```

```
  return($user);
}

sub FindUsersMount {
  # Try to find the mount point for this user
  my $user = shift;
  my $mounts = `$LIST_mount`;
  my ( $mount_point ) = $mounts =~ /\W$user on (\/\S+)\s/;
  return($mount_point);
}

# Remove the indent of a here file... (for debug ouput)
sub herefile {
  my $string = shift;
  $string =~ s/^\s+#.*\n//gm;  # completely remove full line comments
  $string =~ s/#.*//g;         # remove end-of-line comments
  $string =~ s/^\s+\| ?//gm;   # remove the indent part of the line
  $string =~ s/\s+$/\n/g;      # remove any extra end-of-line spaces
  return $string;
}

# ----------------------------------------------------
# Argument Handling and Testing

# Input Argument Variables
my( $user, $mount_point, $unmount );

if ( "$0" =~ /umount/ || $ARGV[0] eq '-u'  ) {
  # Umount Argument handling
  shift if $ARGV[0] eq '-u';
  Usage "Not enough arguments" if @ARGV < 1;
  Usage "Too many arguments"   if @ARGV > 1;
  $_ = shift;
  if ( /\// ) {
    $mount_point = $_;
    $user = FindMountingUser($mount_point) ||
       ErrorExit "Filesystem \"$mount_point\" not mounted";
  }
  elsif ( /^[a-z][a-z0-9]*$/ ) {
    $user = $_;
    $mount_point = FindUsersMount($user) ||
       ErrorExit "User \"$user\" has no H-drive to umount";
  }
  else {  # Invalud pattern
    ErrorExit "Invalud argument \"$_\"";
  }
  $unmount++;
}
else {  # Normal mounting
  Usage "Not enough arguments" if @ARGV < 2;
  $user = shift;
  $mount_point = shift;
  ErrorExit "File system already mounted on \"$mount_point\""
     if FindMountingUser($mount_point);
  # mount options are just ignored at this time
}

ErrorExit "MountPoint \"$mount_point\" is not a path"
   unless $mount_point =~ /^\//;

#ErrorExit "Username \"$user\" is NOT a S-number"
#   unless $user =~ /s[0-9]{4,}/;
```

```
my @user = getpwnam($user);
ErrorExit "User \"$user\" not on local system"  unless @user;
my $mount_link = "$user[7]/H";


# ---------------------------------------------------------------------
# UMount handling

sub CleanUp {
  print "Umount and CleanUp\n" if $DEBUG;
  open(SAVE_OUT, ">&STDOUT"); open(STDOUT, ">/dev/null");
  open(SAVE_ERR, ">&STDERR"); open(STDERR, ">&STDOUT");
  system($CIFS_umount, $mount_point);
  open(STDOUT, ">&SAVE_OUT");
  open(STDERR, ">&SAVE_ERR");

  ErrorExit "File system \"$mount_point\" failed to umount"
     if FindMountingUser($mount_point);

  rmdir $mount_point;

  # Remove the symbolic link - as the user!
  $> = $user[2]  if $< == 0;   # Effectivally become the user!
  unlink $mount_link if -l $mount_link;
  $> = 0   if $< == 0;         # Return to normal root status?
  #system('id');   # check return to root user
}

# Cleanup old mounts   (Umount operation)
CleanUp();
OkExit("Umounted AD H-Drive for User $user") if $unmount;

# ---------------------------------------------------------------------
# ---- STEP 1 ----
# Get password  this is typically not a desiriable thing,
# but as the ldap server does not permit anonymous binds, we don't
# really have a choice!

chomp( my $stty_reset = qx(stty -g 2>/dev/null) );

if ( length $stty_reset ) {  # input from terminal?
  system('stty', '-echo');
  print "Password for $user[6] ($user) H-Drive Mount: ";
}
chomp( my $passwd = <STDIN> );
if ( length $stty_reset ) {
  system('stty', $stty_reset);
  print "\n";
}

#printf "Passwd = \"$passwd\"\n"  if $DEBUG;

# ---- STEP 2 ----
# Retrive Location of Users Home Directory
# If the first server (student) fails, try the other.

my( $AD_Home, $User );
for ( @AD_Types ) {

  $User= lc "$user\@$AD_Domain{$_}";
  print "Get home for $User...\n" if $DEBUG;

  if( $OPTION_RPCCLIENT ) { # switch between "rpcclient" and "ldapsearch"
```

```
  #
  # Get Users homeDirectory simply using samba "rpcclient"
  #
  printf herefile qq{
    |====================================
    |  /usr/bin/rpcclient -U $user $AD_Server{$_} \\
    |    -c 'queryuser $user'
    |------------------------------------
  }   if $DEBUG >= 3;

  # start request for home directory
  my $AD = Expect->spawn(
          '/usr/bin/rpcclient', '-U', $user, $AD_Server{$_},
              '-c', "queryuser $user"
      ) or ErrorExit "rpcclient command failed: $!";

  # Modify communications
  $AD->log_stdout(0)  unless $DEBUG >= 3; # no echo to stdout (unless DEBUG)!
  #$AD->stty(qw( -echo ));    # do not echo input (FAILS)
  #$AD->notransfer(1);        # keep all the output in accumulator (FAILS)

  # Get the output -- give password if requested.
  $AD->expect(10, 'Home Drive',  -re => qr/NT_STATUS_[A-Z_]+/,
                  [ qr/Password:/i =>
                        sub { shift->send("$passwd\n"); exp_continue; } ],
            );
  # let command finish, if not yet done
  $AD->close();

  if( $DEBUG >= 3 ) {
    print "Expect Error: ", $AD->exp_error(), "\n" if $AD->exp_error();
    print "Expect Number: ", $AD->exp_match_number(), "\n";
  }
  # Error Contition from Expect?
  ErrorExit "AD Timeout for $User"
    if $AD->exp_error() == 1; # timeout
  ErrorExit "AD system error $User : $!"
    if $AD->exp_error() == 4; # system error

  # no errors or home directory match - loop
  $AD_Home='';
  next if $AD->exp_error(); # EOF without match

  # Failed to login -- loop
  next if $AD->exp_match() eq 'NT_STATUS_LOGON_FAILURE';

  ErrorExit "AD Failure for $User -- ".$AD->exp_match()
    if ($AD_Home) = $AD->exp_match =~ /NT_STATUS_[A-Z_]+/;

  # gather and check results.
  $AD_Home = (split('\n', $AD->exp_after()))[0];
  $AD_Home =~ s/^[_:\s]+//;
  $AD_Home =~ s/\s+$//;

  ErrorExit "AD Failure for $User -- $AD_Home"
    if $AD->exp_match_number() != 1;  # not the home dir match
}
else {
  #
  # Get Users homeDirectory using LDAP
  # This requires a hardcoded LDAP DN for the users login
  #
  my $LDAP_DN = "CN=$user,$LDAP_DN{$_}";
```

```
  printf herefile qq{
    |===================================
    |  /usr/bin/ldapsearch -x -W -LLL -H \\
    |    ldap://$AD_Server{$_} \\
    |    -D, '$LDAP_DN' \\
    |    -b, '$LDAP_DN' \\
    |    homeDirectory
    |-----------------------------------
  }   if $DEBUG >= 3;

  # start request for home directory
  my $AD = Expect->spawn(
          qw( /usr/bin/ldapsearch -x -W -L -H ),
                'ldap://'.$AD_Server{$_},
                '-D', $LDAP_DN, '-b', $LDAP_DN,
                'homeDirectory'
      ) or ErrorExit "ldapsearch command failed: $!";

  # Modify communications
  $AD->log_stdout(0)  unless $DEBUG >= 3; # no echo to stdout (unless DEBUG)!
  #$AD->stty(qw( -echo ));    # do not echo input (FAILS)
  #$AD->notransfer(1);        # keep all the output in accumulator (FAILS)

  # Get the output -- give password if requested.
  # The 'numEntries:' match is important for fast EOF testing
  $AD->expect(10, 'homeDirectory:', 'ldap_bind:', '# numEntries:',
                  [ qr/Password:/i =>
                      sub { shift->send("$passwd\n"); exp_continue; } ],
            );
  # let command finish, if not yet done
  $AD->close();

  if( $DEBUG >= 3 ) {
    print "Expect Error: ", $AD->exp_error(), "\n" if $AD->exp_error();
    print "Expect Number: ", $AD->exp_match_number(), "\n";
  }

  # Error Contition from Expect?
  ErrorExit "LDAP Timeout for $User"
    if $AD->exp_error() == 1; # timeout
  ErrorExit "LDAP system error $User : $!"
    if $AD->exp_error() == 4; # system error

  # no errors or home directory was matched - loop
  $AD_Home='';
  next if $AD->exp_error(); # EOF without match
  next if $AD->exp_match_number() == 3; # 'numEntries:' home entry not found

  next if $AD->exp_match_number() == 2 &&
          $AD->exp_after() =~ /Invalid credentials/i;

  # Gather and check results.
  $AD_Home = $AD->exp_after();
  $AD_Home =~ s/\n\s+//;     # rejoin lines if needed
  $AD_Home = (split('\n', $AD_Home))[0];
  $AD_Home =~ s/^\s+//;
  $AD_Home =~ s/\s+$//;

  ErrorExit "LDAP Failure for $User -- $AD_Home"
    if $AD->exp_match_number() != 1;  # not the home dir match
}
last if  $AD_Home;   # was a home directory found?
```

```
}

ErrorExit "No AD H-drive found for $user"
   unless $AD_Home;

if ( $DEBUG ) {
  printf "Home found on $User\n";
  printf "AD_Home = $AD_Home\n";
}

# ---- STEP 3 ----
# Determine the Mounting Options for the users H drive mount.

ErrorExit "Unable to get users Type"
    unless $AD_Home =~ /^\W\W([a-z]+)\W/;
my $AD_Type = $1;
printf "AD_Type = $AD_Type\n"      if $DEBUG;

if ( $OPTION_LOCAL_DFS ) {

  # DFS Redirection -- This is a hardcoded hack! Yuck!

  ErrorExit "Unable to get users Campus"
    unless $AD_Home =~ /\\([a-z][a-z])\\/;
  my $AD_Campus = $1;
  printf "AD_Campus = $AD_Campus\n"  if $DEBUG;

  $AD_Home = $AD_DFS{"$AD_Type,$AD_Campus"} . "/$user";
  ErrorExit "Unable to find AD Home Redirection"
    unless defined $AD_Home;

  printf "AD_Home (redir) = $AD_Home\n"  if $DEBUG;
}

# Work out the appropriate CIFS mount options
my $CIFS_options = join( ',',
       "username=$user,uid=$user[2],gid=$user[3]",
       "dir_mode=0700,file_mode=0600",
       "domain=$AD_Domain{$AD_Type}",
     );

# ---- STEP 4 ----
# Mount the Active Directory H-drive

unless ( mkdir($mount_point,0700) ) {
  ErrorExit "mkdir \"$mount_point\" for AD Drive: $!"
    unless $! == 17;  # ignore directory already exists;
}

# Become the user now?  (The mounting cmd SetUID -- for Ubuntu )
ErrorExit "Program \"$CIFS_mount\" not installed"
  unless -x "$CIFS_mount";

# Do the mounting as the user (mount.cifs must be SetUID)
# WARNING: if you do this the user can also umount!
# They will then have a writable directory!
#
#chmod(0700, $mount_point);
#chown($user[2], $user[3], $mount_point ) if $> == 0;
# if ( (stat(_))[2] & 04000 && $> == 0 ) {  # SetUID and we are root!
#   $) = $user[3];   # Effectively become the user!
#   $( = $user[3];
#   $> = $user[2];   # pam_mount will created the mount-point
```

```
#   $< = $user[2];   # and CIFS mount becomes unmountable by the user
# }

printf herefile qq{
  |==================================
  |  $CIFS_mount \\
  |    $AD_Home \\
  |    $mount_point \\
  |    -o $CIFS_options
  |----------------------------------
}   if $DEBUG >= 3;

# start ldap request
my $CIFS = Expect->spawn(
       $CIFS_mount, $AD_Home, $mount_point, '-o', $CIFS_options
     ) or ErrorExit "$CIFS_mount command failed: $!";

# Modify communications
$CIFS->log_stdout(0)  unless $DEBUG >= 3; # no echo to stdout!
#$CIFS->stty(qw( -echo ));    # do not echo input (FAILS)
#$CIFS->notransfer(1);        # keep all the output in accumulator (FAILS)

# Get the output -- give password if requested.
$CIFS->expect(10, -re => qr/error(\(\d+\))?:/,
                [ qr/Password:/i
                      => sub { shift->send("$passwd\n"); exp_continue; } ],
          );

# let command finish, if not yet done
$CIFS->close();

if( $DEBUG >= 3 ) {
  print "Expect Error: ", $CIFS->exp_error(), "\n" if $CIFS->exp_error();
  print "Expect Number: ", $CIFS->exp_match_number(), "\n";
  print "Expect Match: ", $CIFS->exp_match(), "\n";
}


# Error Contition from Expect?
ErrorExit "CIFS Timeout for $User"
  if $CIFS->exp_error() == 1; # timeout
ErrorExit "CIFS system error $User : $!"
  if $CIFS->exp_error() == 4; # system error

my $CIFS_result = (split('\n',$CIFS->exp_after()))[0];
$CIFS_result =~ s/^\s+//;
$CIFS_result =~ s/\s+$//;

printf "CIFS_result = $CIFS_result\n"  if $DEBUG && $CIFS_result;

if ( $CIFS->exp_match() =~ /error/i ) {
  CleanUp();
  ErrorExit "Mount AD Failure -- $CIFS_result"
}
undef $CIFS;

# ---- STEP 5 ----
# Check to make sure all went well.

# Did we actually mount the H-drive?
unless ( FindMountingUser($mount_point) ) {
  CleanUp;     # no mount found!
  ErrorExit "AD Mount Failure -- was not mounted!\n"
```

```
}

# We need to delay a bit before doing a "Not a directory" error test.
# Without this the "ls" succeeds but I have no idea why
sleep(1);

# read test - only capture stderr
my $ls_error = `/bin/sh -c 'ls "$mount_point"' 2>&1 >/dev/null`;
if ( $? != 0 ) {
  CleanUp
  ErrorExit "AD Mount Failure -- Directory Read Failure ($?) $ls_error"
}

print "GOOD MOUNT\n" if $DEBUG;

# Create the link from users home to mount point
# This is done as the user but will NOT cleanup if link fails!

$) = $user[3]  if $< == 0;    # Effectivally become the user!
$> = $user[2]  if $< == 0;
symlink($mount_point, $mount_link)
   or ErrorExit "Mounted H-Drive for $user, Link Failure -- $!";
$> = 0 if $< == 0;            # Return to normal root status?

OkExit("Mounted and Linked AD H-drive for $user");
```