Getting started at TACC

Getting to a remote computer

The Terminal window

- Macs and Linux have a **Terminal** program built-in – find it now on your computer
- Windows 10 or later has **ssh** and **scp** in **Command Prompt** or **PowerShell** (may require latest Windows updates)
    - Open the Start menu Search for **Command**

If your Windows version does not have ssh in **Command Prompt** or **PowerShell**:

- Download the free **PuTTY** suite of remote access tools, which includes a nice Terminal program: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.
    - Either the Putty installer (https://the.earth.li/~sgtatham/putty/latest/w64/putty-64bit-0.77-installer.msi)
    - or just **putty.exe** (Terminal) and **pscp.exe** (secure copy client)

More advanced options for those who want a full Linux environment on their Windows system:

- **Windows Subsystem for Linux** – Windows 10 Professional includes a Ubuntu-like **bash** shells
    - See https://docs.microsoft.com/en-us/windows/wsl/install-win10
    - We recommend the Ubuntu Linux distribution, but any Linux distribution will have an SSH client

From now on, when we refer to "**Terminal**", it is either the Mac/Linux **Terminal** program, Windows **Command Prompt** or **PowerShell**, or the **PuTTY** progr am.

SSH

ssh is an executable program that runs on your local computer and allows you to connect ***securely*** to a remote computer. We're going to use **ssh** to access the **Lonestar6** ***compute cluster*** at TACC, where the remote host name is **ls6.tacc.utexas.edu**.

In your local Terminal window:

SSH to Lonestar6 at TACC

```
ssh <your_TACC_userID>@ls6.tacc.utexas.edu

# For example:
ssh abattenh@ls6.tacc.utexas.edu
```

- Answer ***yes*** to the SSH security question prompt
- Enter the password associated with your TACC account
- Wait for your 2-factor authentication code to arrive via SMS or app, then type it in

If you're using **PuTTY** as your Terminal from Windows:

- Double-click the **Putty** icon
- In the **PuTTY Configuration** window
    - make sure the **Connection type** is **SSH**
    - enter **ls6.tacc.utexas.edu** for Host Name
        - Optional: to save this configuration for further use:
            - Enter **Lonestar6** into the **Saved Sessions** text box, then click **Save**
            - Next time select **Lonestar6** from the **Saved Sessions** list and click **Load**.
    - click **Open** button
    - answer **Yes** to the SSH security question
- In the PuTTY terminal
    - enter your TACC user id after the **"login as:"** prompt, then **Enter**

- enter the password associated with your TACC account
- provide your 2-factor authentication code

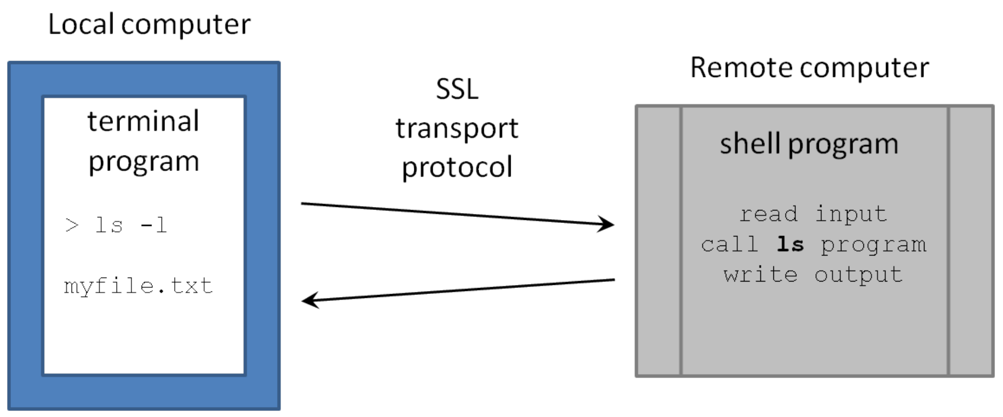
The bash shell

You're now at a ***command line***! It looks as if you're running directly on the remote computer, but really there are two programs communicating:

1. your local Terminal
2. the remote Shell

There are ***many*** shell programs available in Linux, but the default is **bash** (Bourne-again shell).

The Terminal is pretty "dumb" – just sending what you type over its secure sockets layer (SSL) connection to TACC, then displaying the text sent back by the shell. The real work is being done on the remote computer, by programs called by the **bash** shell.



The **bash** command-line environment is extremely powerful, but also complex and unforgiving – a one-character mistake can make all the difference between a command that works and one that doesn't!

In spite of the hurdles, learning to get around the Linux command line will pay substantial dividends. A good place to start is with our Linux fundamentals wiki page.

Setting up your environment

Create some symbolic links and directories

First we will create a few directories and links we will use (more on these later).

You can copy and paste these lines from the code block below into your Terminal window. Just make sure you hit **Enter** after the last line.

Create some ***symbolic links*** that will come in handy later:

Create symbolic directory links

```
cd  # makes your Home directory the "current directory"
ln -s -f $SCRATCH scratch
ln -s -f $WORK work
ln -s -f /work/projects/BioITeam/projects/courses/Core_NGS_Tools CoreNGS
```

Symbolic links (a.k.a. ***symlinks***) are "pointers" to files or directories elsewhere in the file system hierarchy. You can almost always treat a symlink as if it is the actual file or directory.

$WORK and **$SCRATCH** are TACC ***environment variables*** that refer to your **Work** and **Scratch** file system areas. They are like variables in other programming languages, in that they have a name (**WORK**, **SCRATCH**) and hold a value (**$WORK, $SCRATCH**) To see the value of an environment variable, use the **echo** command:

```
echo $SCRATCH
```

The **ln -s** command creates a ***symbolic link***, a shortcut to the linked file or directory.

- Here the link targets are your **Work** and **Scratch** file system areas
- Having these link shortcuts will help when you want to copy files to your **Work** or **Scratch**, and when you navigate the TACC file system using a remote SFTP client
- Always **c**hange **d**irectory (**cd**) to the directory where we want the links created before executing **ln -s**
  - Here we want the links under your home directory (**cd** with no arguments)

Want to know where a link points to? Use **ls** with the **-l** (**l**ong listing) option.

ls -l shows where links go

```
ls -l
```

Set up a **$HOME/local/bin** directory and link several scripts there that we will use in the class.

Set up $HOME/local/bin directory

```
mkdir -p ~/local/bin
cd ~/local/bin
ln -s -f /work/projects/BioITeam/common/bin/launcher_creator.py
```

The tilde (~) character

The tilde character (**~**) is a pathname shortcut that means "Home directory". We'll see more of it later.

$HOME is an environment variable set by TACC that also refers to your **Home** area directory.

Setup your login profile (~/.bashrc)

Now execute the lines below to set up a login script, called **.bashrc**

When you login via an interactive shell, a well-known script is executed to establish your favorite environment settings. We've set up a common login script for you to start with that will help you know where you are in the file system and make it easier to access some of our shared resources. To set it up, perform the steps below:

If you already have a **.bashrc** set up, make a backup copy first.

```
cd
cp .bashrc .bashrc.beforeNGS
```

You can restore your original login script after this class is over.

Copy a pre-configured login script

```
cd
cp /work/projects/BioITeam/projects/courses/Core_NGS_Tools/tacc/bashrc.corengs.ls6 .bashrc
chmod 600 .bashrc
```

What's going on with **chmod**?

- The **chmod 600 .bashrc** command marks the file as readable and writable ***only by you***.
  The **.bashrc** script file will not be executed unless it has these ***exact*** permissions settings.
- The well-known filename is **.bashrc** (or **.profile** on some systems), which is specific to the **bash** shell.

Since **.bashrc** is executed when you login, to ensure it is set up properly you should first log off **ls6** like this:

Log off Lonestar6

```
exit
```

Then log back in to **ls6.tacc.utexas.edu**. This time your **.bashrc** will be executed and you should see a new shell prompt:

```
ls6:~$
```

The great thing about this prompt is that it always tells you where you are, which avoids you having to execute the **pwd** (**p**resent **w**orking **d**irectory) command every time you want to know where you are. Execute these commands to see how the prompt reflects your current directory.

```
mkdir -p ~/tmp/a/b/c
cd ~/tmp/a/b/c

# Your prompt should look like this:
ls6:~/tmp/a/b/c$
```

The prompt now tells you you are in the **c** sub-directory of the **b** sub-directory of the **a** sub-directory of the **tmp** sub-directory of your **Home** directory (**~**).

So why don't you see the **.bashrc** file you copied to your home directory? Because all files starting with a period (***dot files***) are hidden by default. To see them add the **-a** (**a**ll) option to **ls**:

How to see hidden files

```
cd
ls -a
```

To see even more detail, including file type and permissions and symbolic link targets, add the **-l** (**l**ong listing) option:

Long listing form of ls

```
ls -la
```

ll alias

Your new **~/.bashrc** file defines a **ll** alias command, so when you type **ll** it is short for **ls -la**.

Details about your login script

We list the contents of your **.bashrc** login script to the Terminal with the **cat** (con**cat**enate files) command. **cat** simply reads a file and writes each line of content to ***standard output*** (here, your Terminal):

List .bashrc file contents without pausing

```
cd
cat .bashrc

# or for larger files...
more .bashrc
```

Don't use cat for large files

The **cat** command just displays the entire file's content, line by line, without pausing, so should not be used to display large files. Instead, use a ***pager*** (like **more** or **less**) or look at parts of the file with **head** or **tail**.

You'll see the following (you may need to scroll up a bit to see the beginning):

Contents of your .bashrc file

```
#!/bin/bash
# TACC startup script: ~/.bashrc version 2.1 -- 12/17/2013
#   This file is NOT automatically sourced for login shells.
# Your ~/.profile can and should "source" this file.
# Note neither ~/.profile nor ~/.bashrc are sourced automatically
# by bash scripts.
#   In a parallel mpi job, this file (~/.bashrc) is sourced on every
# node so it is important that actions here not tax the file system.
# Each nodes' environment during an MPI job has ENVIRONMENT set to
# "BATCH" and the prompt variable PS1 empty.
#################################################################
# Optional Startup Script tracking. Normally DBG_ECHO does nothing
if [ -n "$SHELL_STARTUP_DEBUG" ]; then DBG_ECHO "${DBG_INDENT}~/.bashrc{"; fi
##########
# SECTION 1 -- modules
if [ -z "$__BASHRC_SOURCED__" -a "$ENVIRONMENT" != BATCH ]; then
  export __BASHRC_SOURCED__=1
  module load launcher
fi
############
# SECTION 2 -- environment variables
if [ -z "$__PERSONAL_PATH__" ]; then
  export __PERSONAL_PATH__=1
  export PATH=.:$HOME/local/bin:$PATH
fi
# For better colors using a dark background terminal, un-comment this line:
#export LS_COLORS=$LS_COLORS:'di=1;33:fi=01:ln=01;36:'
# For better colors using a white background terminal, un-comment this line:
#export LS_COLORS=$LS_COLORS:'di=1;34:fi=01:ln=01;36:'
export LANG="C"  # avoid the annoying Perl locale warnings
export BIWORK=/work/projects/BioITeam
export CORENGS=$BIWORK/projects/courses/Core_NGS_Tools
export BI=/corral-repl/utexas/BioITeam
export ALLOCATION=OTH21164        # For ls6        Group is G-824651
##export ALLOCATION=UT-2015-05-18 # For stampede2  Group is G-816696

##########
# SECTION 3 -- controlling the prompt
if [ -n "$PS1" ]; then PS1='ls6:\w$ '; fi
##########
# SECTION 4 -- Umask and aliases
#alias ls="ls --color=always"
alias ll="ls -la"
alias lah="ls -lah"
alias lc="wc -l"
alias hexdump='od -A x -t x1z -v'
umask 002
##########
# Optional Startup Script tracking
if [ -n "$SHELL_STARTUP_DEBUG" ]; then DBG_ECHO "${DBG_INDENT}}"; fi
```

So what does this login script do? A lot! Let's look at just a few of them.

the "she-bang"

The first line is the ***she-bang***. Even though the expression is inside a shell comment (denoted by the **#** character), it tells the shell (**bash**) what program should execute this file – in this case, **bash** itself.

The "she-bang" line

```
#!/bin/bash
```

environment variables

The login script also sets an environment variable **$BIWORK** to point to the shared directory **/work/projects/BioITeam**, and another environment variable **$CORENGS** to point to the specific sub-directory for our class.

Setting environment variables to useful locations

```
export BIWORK=/work/projects/BioITeam
export CORENGS=$BIWORK/projects/courses/Core_NGS_Tools
```

Environment variables are like variables in a programming language like **python** or **perl** (in fact **bash** is a complete programming language). They have a name (like **BIWORK** above) and a value (the value of **$BIWORK** is the pathname **/work/projects/BioITeam**). Read more about environment variables here: More on environment variables.

shell completion

You can use these environment variables to shorten typing, for example, to look at the contents of the shared **/work/projects/BioITeam** directory as shown below, using the magic **Tab** key to perform shell completion.

Important Tip -- the Tab key is your BFF!

The **Tab** key is one of your best friends in Linux. Hitting it invokes ***shell completion***, which is as close to magic as it gets!

- **Tab** once will expand the current command line contents as far as it can unambiguously.
  - if nothing shows up, there is no unambiguous match
- **Tab** twice will give you a list of ***everything*** the shell finds matching the current command line.
  - you then decide where to go next

Shell completion exercise

```
# hit Tab once after typing $BIWORK/ to expand the environment variable
ls $BIWORK/

# now hit Tab twice to see the contents of the directory
ls /work/projects/BioITeam/

# type "pr" and hit Tab again
ls /work/projects/BioITeam/pr

# type "co" and hit Tab again
ls /work/projects/BioITeam/projects/co

# type "Co" and hit Tab again
ls /work/projects/BioITeam/projects/courses/Co

# your command line should now look like this
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/

# now type "mi" and one Tab
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/mi

# your command line should now look like this
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/misc/

# now hit Tab once
# the shell expands as far as it can unambiguously,
# so your command line should look like this
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/misc/small

# now hit Tab twice
# You should see 3 filenames, all starting with "small"
# small.bam  small.fq   small2.fq

# type a period (".") then hit Tab twice again
# You're narrowing down the choices -- you should see two filenames
```

```
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/misc/small
# small.bam  small.fq

# finally, type "f" then hit Tab again. It should complete to this:
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/misc/small.fq
```

extending the $PATH

When you type a command name the shell has to have some way of finding what program to run. The list of places (directories) where the shell looks is stored in the **$PATH** environment variable. You can see the entire list of locations by doing this:

See where the bash shell looks for programs

```
echo $PATH
```

As you can see, there are a lot of locations on the **$PATH**. That's because when you load modules at TACC (such as the **module load** lines in the common login script), that makes the programs available to you by putting their installation directories on your **$PATH**. We'll learn more about modules later.

Here's how the common login script adds your **$HOME/local/bin** directory to the location list – recall that's where we linked several useful scripts – along with a special dot character (`.`) that means "here", or "whatever the current directory is". In the statement below, colon (`:`) separates directories in the list.

Adding directories to PATH

```
export PATH=.:$HOME/local/bin:$PATH
```

setting up the friendly command prompt

The complicated looking **if** statement in SECTION 3 of your **.bashrc** sets up a friendly shell prompt that shows the current working directory. This is done by setting the special **PS1** environment variable and including a special **\w** directive that the shell knows means "current directory".

Setting up the friendly shell prompt for stampede

```
##########
# SECTION 3 -- controlling the prompt
if [ -n "$PS1" ]; then PS1='ls6:\w$ '; fi
```