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: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html.
  - Either the Putty installer 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** program.

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 (Texas Advanced Computing Center), 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
  - this will only be asked the 1st time you access **ls6**
- Enter the password associated with your TACC account
  - for security reasons, your password characters will not be echoed to the screen
- Get your 2-factor authentication code from your phone's **TACC Token** app, and 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

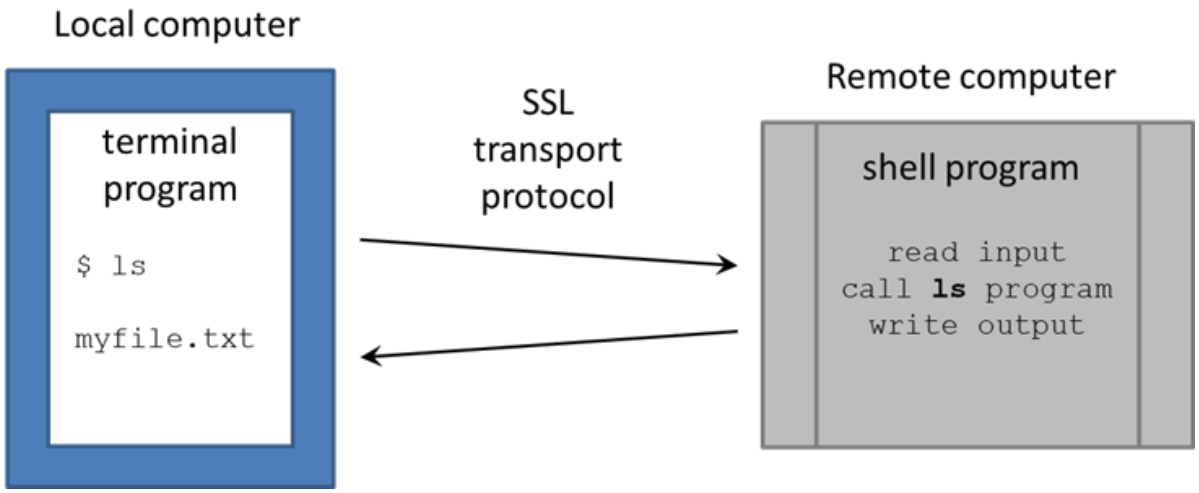
The bash shell

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** (**B**ourne-**a**gain **sh**ell).

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 ***executable programs*** called by the **bash** shell (also called ***commands***, since you call them on the command line).



About the command line

Read more about the command line and commands on our **Linux fundamentals** page:

- **The bash shell REPL and commands**
- **Getting help**
- **Literal characters and metacharacters**
- **About command line input**

Setting up your environment

Setup your login profile (~/.bashrc)

Now execute the lines below to set up a login script, called **~/.bashrc**. [Note the **tilde** (**~**) is shorthand for "***my Home directory***". See **Linux fundamentals: pathname syntax**]

When you login via an interactive shell, a well-known script is executed to establish your favorite environment settings. The well-known filename is **~/.bashrc** (or **~/.profile** on some systems), which is specific to the **bash** shell.

We've pre-created a common login script for you 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:

> 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.

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

```
cd
ls -la
# Do you see a .bashrc file? If so, save it off
cp .bashrc .bashrc.beforeNGS
```

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

If your **Terminal** has a ***dark background*** (e.g. black), copy this file:

Copy a pre-configured login script for dark background Terminals

```
cp /corral-repl/utexas/BioITeam/core_ngs_tools/login/bashrc.corengs.ls6.dark_bg  ~/.bashrc
chmod 600 ~/.bashrc
```

If your **Terminal** has a ***light background*** (e.g. white), copy this file:

Copy a pre-configured login script for light background Terminals

```
cp /corral-repl/utexas/BioITeam/core_ngs_tools/login/bashrc.corengs.ls6.light_bg  ~/.bashrc
chmod 600 ~/.bashrc
```

So why don't you see the **.bashrc** file you just copied when you do **ls**? Because all files starting with a period (***dot files***) are hidden by default. To see them add the **-l** (**l**ong listing) and **-a** (**a**ll) options to **ls**:

```
# show a long listing of all files in the current directory, including "dot files" that start with a period
ls -la
```

(Read more about **File attributes**)

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.

Since your **~/.bashrc** is executed when you login, to ensure it is set up properly you should first log off **Lonestar6** like this:

Log off Lonestar6

```
exit
```

Your **Terminal** has logged off of **Lonestar6** and is back on your local computer.

Now log back in to **ls6.tacc.utexas.edu**. This time your **~/.bashrc** will be executed to establish your environment:

ll alias

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

You should see a new ***command line 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 what the current directory is. Execute these commands to see how the prompt reflects your current directory.

```
# mkdir -p says to create all parent directories in the specified path
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 (**~**).

Your login script has configured this command prompt behavior, along with a number of other things.

Create some symbolic links and directories

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 -sf /work/projects/BioITeam/projects/courses/Core_NGS_Tools CoreNGS

ls # you'll see the 3 symbolic links you just created
```

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 – more on these file system areas soon. (Read more about **Environment variables**)

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 **~/local/bin** directory and link a script there that we will use in the class.

Set up ~/local/bin directory

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

Since our **~/.bashrc** login script added **~/local/bin** to our **$PATH**, we can call any script or command in that directory with just its file name. And Tab completion works on program names too:

```
cd

# hit Tab once after typing "laun"
# This will expand to launcher_creator.py
```

Details about your login script

Let's take a look at the contents of your **~/.bashrc** login script, using the **cat** (con**cat**enate files) command. **cat** simply reads a file and writes each line of content to ***standard output*** (here, your **Terminal**):

Display .bashrc file contents

```
cd
cat .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**. For example:

more ~/.bashrc

This will display one "***page***" (Terminal screen) of text at a time, then pause. Press **space** to advance to the next page, or **Ctrl-c** to exit **more**.

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
```

There's a lot of stuff here; let's look at just a few things.

Environment variables

The login script sets several ***environment variables***.

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 other programming languages 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 of the shared **/work/projects/BioITeam** directory).

To see the ***value*** of an environment variable, use the **echo** command, then the variable name after a **dollar sign** ($):

```
echo $CORENGS
```

We'll use the **$CORENGS** environment variable to avoid typing out a long pathname:

```
ls $CORENGS
```

(Read more about **Environment variables**)

Shell completion with Tab

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

Follow along with this:

Shell completion exercise

```
# hit Tab once to expand the environment variable name
ls $BIW

# hit Tab again 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
# There is no unambiguous match, so hit Tab again
# After hitting Tab twice you should see several filenames:
# fastqc/ small.bam  small.fq   small2.fq

# now type "sm" and one Tab
# your command line should now look like this
ls /work/projects/BioITeam/projects/courses/Core_NGS_Tools/misc/small

# 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**.

Here's how the common login script adds the **~/local/bin** directory you created above, to the location list, 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. (Read more about **pathname syntax**)

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
```