

Linguistics Computing Resources

Brandon Graves

Department of Linguistics
University of Washington

September 22, 2022

Contact information

How to get in touch if you encounter problems

Introduction

CompLing Resources

- Treehouse Lab
- Treehouse Lab Policies
- Corpora database
- GitLab Server
- Prime Directive
- Shell access
- Python
- Filesystem access
- Data protection
- Data security
- Intro to Condor
- Job Requirements
- Advanced Condor
- Condor commands
- Condor troubleshooting

Further reading

For Problems & Requests: lingit@u.washington.edu
For Questions: linghelp@u.washington.edu
Padelford B-5-G (Floor PL)

Online copies of this information

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

- PDF copy of these slides:
 `> /opt/dropbox//orientation/welcome_to_patas.pdf`
- CompLing wiki: <https://wiki.ling.washington.edu/>

Introduction

CompLing Resources

Treehouse Lab

Treehouse Lab
Policies

Corpora database

GitLab Server

Prime Directive

Shell access

Python

Filesystem access

Data protection

Data security

Intro to Condor

Job Requirements

Advanced Condor

Condor commands

Condor

troubleshooting

Further reading

- Four Linux workstations
- Log in with your CLMS account
- Access same file resources as the computing cluster, including home directories.
- Can run Windows 7 in a VM
- Lock PIN codes will be emailed to by Request. Pleast contact me if you would like access to the lab.

Treehouse Lab policies

Introduction

CompLing Resources

Treehouse Lab

**Treehouse Lab
Policies**

Corpora database

GitLab Server

Prime Directive

Shell access

Python

Filesystem access

Data protection

Data security

Intro to Condor

Job Requirements

Advanced Condor

Condor commands

Condor

troubleshooting

Further reading

- Keep the door closed
- Covered drinks only
- Close the window if you're the last to leave.
- **No food in the lab**

CompLing database

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies

Corpora database

GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

Located at <https://cldb.ling.washington.edu>

Shows:

- Corpora we have on hand & whether/Where they're currently installed
- Linguistics software installed on the cluster
- Job postings

GitLab Server

<https://git.ling.washington.edu/>

- You must sign up with your UW email. Non UW email addresses are blacklisted.

Prime Directive

- Do not copy/paste commands from the internet.
- You do not have "sudo" access
- "sudo" is a red flag!
- email me instead if you need software installed
- <https://explainshell.com> is good! Check your commands here!

Shell access

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies

Corpora database

GitLab Server

Prime Directive

Shell access

Python

Filesystem access

Data protection

Data security

Intro to Condor

Job Requirements

Advanced Condor

Condor commands

Condor

troubleshooting

Further reading

- SSH to patas.ling.washington.edu or dryas.ling.washington.edu
- A link with more information and suggested SSH clients will be provided in your account creation email.
- Linguistics software installed under `/NLP_TOOLS`
- Commonly used software(python, java, etc..) located under: `/opt`
- Corpora under `/corpora`

Request an account at

<https://cldb.ling.washington.edu/live/accountrequest-form.php>

Introduction

CompLing Resources

Treehouse Lab

Treehouse Lab
Policies

Corpora database

GitLab Server

Prime Directive

Shell access

Python

Filesystem access

Data protection

Data security

Intro to Condor

Job Requirements

Advanced Condor

Condor commands

Condor

troubleshooting

Further reading

- Python 2.X is begin retired. Use Python 3.x wherever possible
- Using Python 3.6 is better than using 3.4 or lower!

Filesystem access

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

- SCP or SFTP to patas or dryas — best option from off campus.
- use WinSCP application to transfer files on windows.

Data protection

Protecting your data from loss

- File servers use redundant disk arrays (RAID)
- All servers are backed up nightly.
- Contact lingit@u.washington.edu if you need data restored from backup.
- **Backups are not a guarantee** — you should retain your own copies of data you cannot afford to lose.
- More information: See the DataProtection wiki page.

Data security

Keeping your data private

- Patas cluster:
 - By default, home directories are readable by everyone.
 - If that isn't what you want, `chmod og-rx $HOME`
 - You can also do this just to individual subdirectories that you want to keep private.

Introduction to Condor

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

Condor is a **batch-oriented** clustering system. It's the more general-purpose of the two major parallel computing systems we support on our cluster (the other being Hadoop.)

- Jobs are submitted to a queue and matched with an available computer
- Jobs are run non-interactively
- A **submit description file** is used to tell Condor how to run the job.
- Input and output are directed to files

A quick review of Unix standard I/O

stdin, stdout, and stderr

- **stdin**
 - Connected to the keyboard when a command is run interactively.
 - Can be re-directed from a file with the < operator:
`mycommand <myinput.txt`
- **stdout**
 - Connected to the screen when a command is run interactively.
 - Can be re-directed to a file with the > operator:
`mycommand >myoutput.txt`
- **stderr**
 - Used to for error messages and diagnostics, so they don't disappear if output is redirected.
 - Connected to the screen when a command is run interactively.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
```

```
getenv = true
```

```
input = text.in
```

```
output = results.out
```

```
error = wc.error
```

```
log = wc.log
```

```
notification = complete
```

```
arguments = "-w"
```

```
request_memory = 512
```

```
request_GPUs = 1
```

Queue

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

A simple example

For a command we can run as:

```
wc -w <text.in >results.out
```

The submit description file might look like this:

```
executable = /usr/bin/wc
getenv = true
input = text.in
output = results.out
error = wc.error
log = wc.log
notification = complete
arguments = "-w"
request_memory = 512
request_GPUs = 1
Queue
```

Tip: Sometimes it's easier to write a shell script to run your command, then use the script as the executable. This also makes testing easier.

How Condor runs a job

- User submits job with `condor_submit`:

```
condor_submit wc.cmd
```

Note: This must be done from patas or dryas, not from a Treehouse workstation.

- Condor adds job to queue
- When a matching machine is available, the job is executed there
- User is notified via email when job completes (username@u.washington.edu; use `notify_user` to override, `notification=Never` to disable)

Job Requirements

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

- Condor allows you to specify how much memory your job needs
- Use the `request_memory` option; value is in megabytes
- Default is 2048 megabytes
- If you guess low your job may be evicted; if you guess high you needlessly limit which machines can run your job.
- The **SIZE** column in `condor_q` shows you how much memory your job is currently using

Advanced Condor usage

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

- Multiple jobs can be launched from the same submit description file, with different files and arguments
- See the wiki and `/condor/examples` to see how

Whenever possible, break long-running jobs up into multiple chunks that can be run in parallel, and queue them all simultaneously. This lets you use many CPUs instead of one or two.

An advanced example

Run `mycommand` on 10 files, named `mycommand.in0` through `mycommand.in9`:

```
Executable = mycommand
input      = mycommand.in$(Process)
output     = mycommand.out$(Process)
error      = mycommand.error$(Process)
Log        = mycommand.log
arguments  = "-a -n"
Queue 10
```

- May have multiple Queue lines, with any settings you want to change listed between them
- For complex jobs, consider writing a program to generate the submit file

Research job tracking

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

- We track the percentage of the cluster used by research jobs, to help qualify our program for a research sales tax exemption.
- To help, add `+Research=true` to your submit description file when you run research-related jobs. Do not use this for classwork, etc.
- This does not affect job scheduling; it is only for recordkeeping.

Some useful Condor commands

Introduction

CompLing Resources

Treehouse Lab
Treehouse Lab
Policies
Corpora database
GitLab Server
Prime Directive
Shell access
Python
Filesystem access
Data protection
Data security
Intro to Condor
Job Requirements
Advanced Condor
Condor commands
Condor
troubleshooting

Further reading

- `condor_submit` — submit a job
- `condor_status` — list available nodes and their status
- `condor_q` — list the job queue
- `condor_rm` — remove a job from the queue

Condor troubleshooting

What to do if it doesn't work

- Check the job log file for clues about what's going on.
- Job sits in queue — use `condor_q -analyze [jobid]` to see why your job isn't being matched with a node.
- Job gets held — use `condor_q -long [jobid]` and look at the `HoldReason` parameter.
- Double-check your arguments and input files — run the executable on the command line to test.
- If your executable isn't in the directory you're submitting from, did you supply the full path?
- See the [TroubleshootingCondor](#) page of the UWCL Wiki.
- If all else fails, email lingit@u.washington.edu Leave the job in the queue so I can look at it. Attaching the job log file is also helpful.

Further reading

- **CompLing Wiki:**
<http://wiki.ling.washington.edu/>
- **Manual pages:**
`man condor_submit`, `man condor_q`, **etc.**
- **Official Condor manual:**
<http://www.cs.wisc.edu/condor/manual/v7.6/>