Quick/practical references for common tasks

Coding

Bash

Command line parsing: quick and dirty

```
#!/bin/bash
echo "total number of arguments is ${#@}"  ## does not include the script name
echo "script name is $0"
echo "first argument is $1"
echo "second argument is $2"
#...
```

Command line parsing: getopts

- https://wiki.bash-hackers.org/howto/getopts_tutorial
- Basic example

```
function test1 () {
    function func_usage() { echo "Usage: test1 -ac -b <parameter>"; return
0 ; }

    local OPTIND opt a b c  ## Only needed inside function
    while getopts "ab:c" opt; do ## -a and -c are flags, -b is option with
argument
        case "$opt" in
                a)
                echo "-a triggered"
                ;;
                b)
                echo "setting local variable b"
                local b=$OPTARG  ##OPTARG stores the flag or unknown option
                echo "local b is $b"
                ;;
                c)
                echo "-c triggered"
                ;;
                \?)   ## What is \? ? this gets triggered with by undefined
flag
                func_usage
                return 1
                ;;
                :)  ## this gets triggered by option missing its argument
```

```
                func_usage
                return 1
                ;;
        esac
    done
    return 0
}
```

Basic loop syntax

- Looping over arrays

```
arr1= ("a" "b" "c")
## Method 1
for i in ${arr1[@]}
    do
    echo $i
done
## Method 2 - advantage: can loop over multiple arrays with same index
for (( i=0; i<${#arr1[@]}; i++ ))  ## itialize, test, update
        do
        echo $i
done
```

- While loop: TODO write this

String trimming

- Prefixes and suffixes

```
x=prefix_xyxyx_suffix
echo ${x#prefix_} ## removes prefix ->xyxyx_suffix
echo ${x%_suffix}  ## remove suffix -> prefix_xyxyx
```

- file extensions and basenames

```
fname=/path/to/myfile.ext
filename=$(basename $fname)  ## myfile.ext
extension="${filename##*.}" ## ext
filename="${filename%.*}"  ## /path/to/myfile
```

Prepend/append to arrays

```
array=( "${array[@]/#/prefix_}" )
array=( "${array[@]/%/_suffix}" )
```

Default variable values

- https://unix.stackexchange.com/questions/122845/using-a-b-for-variable-assignment-in-scripts/122878

```
a="${:-default}"  ## if a undefined or empty string then set to default.
Otherwise unchanged
```

Named arrays

- TODO: write this

Redirection of multiple input streams

- TODO: write this

Python

Command line parsing: sys.argv

```
import sys
print( sys.argv[0])  ## script name (sys.argv is just a list)
print( sys.argv[1])  ## First string after script name
print(sys.argv[2])   ## Second string after script name
```

Comman line parsing: argparse

- TODO: basic example

Matplotlib

- setting good color schemes: TODO write this
- good matplotlib.rc file : TODO add this as a separate document

R

Read file into array

```
arr <- scan(file , what = "character")
```

Write array

```
write(myArr , file="myFname", sep = "\n")
```

Command line parsing: quick and dirty

```
args = commandArgs(trailingOnly = TRUE ) ## args is a character vector
print(args[1]) ## arg1 when invoked as Rscript myScript.R arg1 arg2
print(args[2]) ## arg2 when invoked as Rscript myScript.R arg1 arg2
#...
```

Command line parsing: argparser library

```
library("argparser")
#####################################################
#### PARSE
p <- arg_parser(description = "My description")
p <- add_argument(p, "--param1", default= "optional", type = "default is 'string'
another option is numeric" , help = "Must have help or will throw error" )
p <- add_argument(p , "--flag1" , flag= TRUE, help = "" )  ## this is a flag
p <- add_argument(p , "param2" , help = "")
# ...
argv <- parse_args(p) ## this will parse from the trailing arguments to scrip at
cmd line
## For debug you might want to pass arguments inside an R session and not from
cmd line:
## To this using
## argv <- parse_args(p, c("--param1" , "param1_value", "--flag1" , "--param2"
"param2_value"))

#####################################################
#### Using the argv oject
## Accessing parameter arguments
argv$param1
## Testing flags
if (args$flag1){
    ## do something
}
## Testing if parameter is unspecified
if ( is.na(argv$param2) ){
    print("param2 value not specified")
}
```

Install package locally

1. Download the .tar.gz file for package from https://cran.r-project.org/

2. Run command:

```
R CMD INSTALL -l <My/local/lib> <pkgName>.tar.gz
```

3. To import library into R script use

```
library("<pkgName>" , lib.loc="<My/local/lib>")
```

Installing multiple version on same machine

- https://irvingduran.com/2016/10/installing-multiple-version-of-r-on-the-same-machine-for-macos-mac/

AWK

If/else

```
awk -F '{ if ( <condition> ) {<action1>; <action2> } else { <action> } }' <f_in>
```

passing external variables

```
var1=2
var2=4
awk -v x=$var1 -v y=$var2 '$2 == x {print y " " $1}' <f_in>
```

printf

```
awk '{printf "%s\t%s\t%s\n" , $1 , $2 , $3}'
```

Biocluster / SLURM

Setup local install directory and `pip install --user <packageName>`

- Setup (Only need to do this once)
    1. Create a `.local` directory for user-specific python packages:

```
mkdir /home/my/prefered/dir/.local
```

2. Set `PYTHONUSERBASE` environmental variable by adding the following line to your .bashrc :

   ```
   export PYTHONUSERBASE="/home/my/prefered/dir/.local"
   ```

   then,

   ```
   source ~/.bashrc
   ```

- Installing your packages
  - With your preferred version of python loaded run

    ```
    pip install --user <MyPackageName>
    ```

`sbatch` - basic

```
sbatch -p <queue> --cpus-per-task=1 --mem=12GB -D <workDir> --job-name=<jobName>
-o "oe/<jobName>.o" -e "oe/<jobName>.e" --export=var1=$var1,var2=$var2...
PATH/TO/SCRIPT/example.slurm
```

- `var1` and `var2` are passed to example.slurm

`sbatch` - without slurm script

```
sbatch -p <queue> --cpus-per-task=1 --mem=12GB -D <workDir> --job-name=<jobName>
-o "oe/<jobName>.o" -e "oe/<jobName>.e" <<EOF
#!/bin/bash
script_line1 ##lines will be submitted by slurm just like they were in a script
script_line2
...
EOF
```

Jupyter notebooks

Use the full window

- Execute the following line in a cell

  ```
  from IPython.core.display import display, HTML
  display(HTML("<style>.container { width:100% !important; }</style>"))
  ```

- Or, add the following line to the file `~./jupyter/custom/custom.css`:

```
.container { width:100%: !important; }
```

Git and githib

Push to remote repository (remote repository URL already set)

1. Stage (adds files in local repository to set of staged files)

```
git add .   ## To unstage a file use:  git reset HEAD <YOUR-FILE>
```

2. Commit

```
git commit -m <commit message>
```

3. Push

```
 git push -u origin master
```

Check if local repository is up-to-date

- https://stackoverflow.com/questions/7938723/git-how-to-check-if-a-local-repo-is-up-to-date)

```
git remote show origin
# Returns something like:
#HEAD branch: master
#  Remote branch:
#    master tracked
#  Local branch configured for 'git pull':
#    master merges with remote master
#  Local ref configured for 'git push':
#    master pushes to master (local out of date)  ##<-------
```

Force git pull to overwrite local files

- https://stackoverflow.com/questions/1125968/how-do-i-force-git-pull-to-overwrite-local-files

```
git fetch --all
git reset --hard origin/<branchName>  ## branchName is probably master
```

References

- TODO: add some good ones (concise)

Markdown

Markdown is a simple language that lets you write text, tables, code blocks (syntax highlighting), math (latex syntax) and more. Markdown files have .md extension

- A good code editor for markdown is Typora (https://typora.io/)
- A good reference for the markdown language is https://support.typora.io/Markdown-Reference/

Misc

tar/untar directory

- Tar

  `tar -zcvf archive-name.tar.gz directory-name`

  - see https://www.cyberciti.biz/faq/how-do-i-compress-a-whole-linux-or-unix-directory/
- Untar
  - TODO: add example