Linux Programming, Git and Github

Git and Github

- Version control
- Store your code and share with others
- Team collaboration

Github

- Go to Github and register an account: https://github.com/
- You can use Github in two ways: use command or not. In the following steps, you will try both ways.

1. Create your first repository in Github

- Follow all steps on this page: https://guides.github.com/activities/hello-world/
- You'll need to finish the following:
  - Create a Repository
  - Create a Branch
  - Make a Commit
  - Open a Pull Request
  - Merge Pull Request

2. Command line version control - Modify existing repository

- 1) Sign up on your machine

  ```
  git config --global user.name "<your_name_here>"
  ```

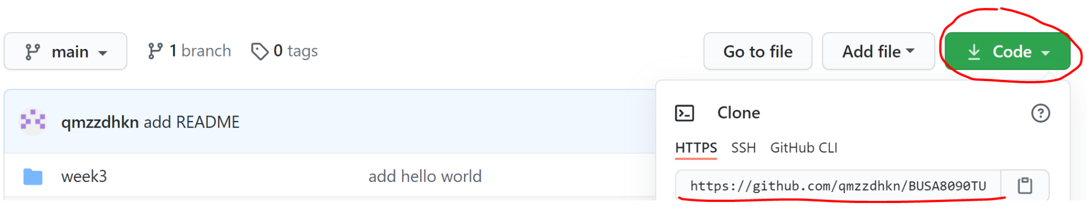
- 2) Tell Git your email, and make sure it's the same email you used when you signed up for GitHub

  ```
  git config --global user.email "<your_email@email.com>"
  ```

- 3) Make a copy of other people's repository to your own computer.

  ```
  git clone <git repository URL>
  ```

  For the URL, go to the repository you just created --> CODE, and copy the address



- 4) Go to the cloned folder

  `cd <Your cloned folder>`

- 5) A good practice is to AVOID directly working on the 'main' branch. This will mess up things for you when you want to merge your changes. Create a new branch, checkout the new branch, work in this branch.

  `git checkout -b <new_branch_name>` # create new branch and checkout

- 6) Always make sure your branch is up to date before you make any changes

  `git pull origin main`

- 7) Check you file status: `git status`

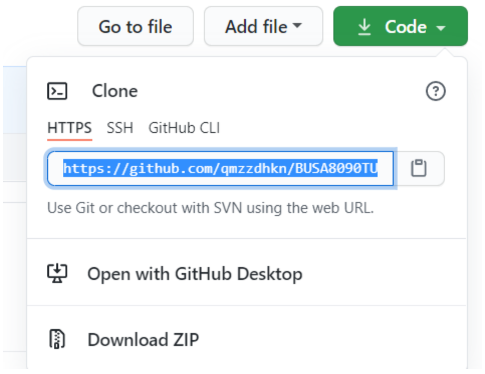
  git has three areas:

  - working directory (real folder)
  - staging area (your change immediately added here from directory before commit)
  - commit --> repository

- 8) Track file(s) in staging area:
  - add all files in the folder `git add .`
  - add single file `git add <filename>`
- 9) Make some changes now
- 10) Check the current status: `git status`
- 11) Commit changes: `git commit -m "<add description of changes made>"`
- 12) Check the current status again: `git status`
- 13) Push your changes to Github
  - `git push origin <current branch name>`
- 14) Now go to Github to check your pull request
- 15) If everything is okay, you can merge the changes to your original files
  - create a pull request
  - review and confirm the merge
  - go back to your repository files, you will find you files are changed

3. Command line version control - Create a new repository

- 1) Create a new empty repository on Github, this time, DO NOT initiate with a 'Readme' file.
- 2) Go to the folder you want to share on Github and initiate a Git repository: `git init` (If the current folder is a clone from Github, this step is not necessary)
- 3) Check you file status: `git status`
- 4) Track file(s) in staging area:
  - add all files in the folder `git add .`
  - add single file `git add <filename>`
- 5) Check the current status: `git status`
- 6) Commit changes: `git commit -m "<add description of changes made>"`
- 7) Check the current status again: `git status`
- 8) Publish your files on Github:
  - 1. `git remote add origin <your empty Github repository URL, see below screenshot>`
  - 2. `git push origin main`



https://github.com/qmzzdhkn/BUSA8090TU

Use Git or checkout with SVN using the web URL.

Linux Programming

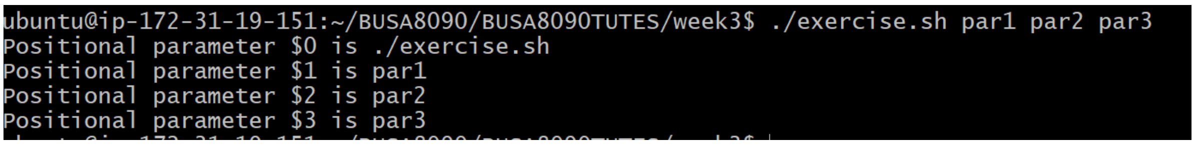
- Variables, e.g. `$HOME`, `$PWD`
- Parameters, `$1 - $9`, `$#`, `$*`, `$@`, `$0`
- Commands, e.g. `echo`, `grep`
- Flow Controls:
  - `if...then...elif...else...fi`
  - `test`
  - `while...do...done`
  - `until...do...done`

- `for...in...do...done`
- `case...in...esac`
- `select...in...do...done`

We'll try some examples.

Parameters

```
#!/bin/bash
# This file is saved as 'exercise.sh'
echo "Positional parameter \$0 is $0"  # what is the meaning for $0?
\$0?
x=1
for i in "$@"  # usage of control flow
do
echo "Positional parameter \$$x is $i"  # what is the meaning of \$$x?
$i?
x=$(expr $x + 1)  # what does this line do?
done
```



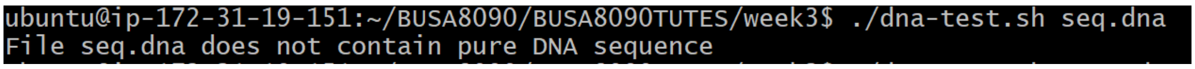
```
ubuntu@ip-172-31-19-151:~/BUSA8090/BUSA8090TUTES/week3$ ./exercise.sh par1 par2 par3
Positional parameter $0 is ./exercise.sh
Positional parameter $1 is par1
Positional parameter $2 is par2
Positional parameter $3 is par3
```

Check for DNA as File Content

```
#!/bin/bash
# save as dna-test.sh
# test if file contains dna sequence
if test -z "$(cat $1)" && test -f $1; then
# what is $1?
# what does '&&' mean? what did the two options of test command '-z'
and '-f' do?
# What did "$(cat $1)" do?
        echo "File $1 is empty"
        exit
fi
grep -sq '[^acgt]' $1  # what is the purpose of this line?
result=$?  # what is $?
if test $result -eq 0 ; then  # what is the test?
        echo "File $1 does not contain pure DNA sequence"
elif test $result -eq 1; then
        echo "File $1 does contain pure DNA sequence"
elif [ $result -eq 2 ]; then  # what does [ $result -eq 2 ] mean?
        echo "File $1 does not exit"
else
```

```
        echo "Some error occured!"
fi
```

run `./dna-test.sh seq.dna`



```
ubuntu@ip-172-31-19-151:~/BUSA8090/BUSA8090TUTES/week3$ ./dna-test.sh seq.dna
File seq.dna does not contain pure DNA sequence
```

Time Signal

```
#!/bin/bash
# save as time-signal.sh
# gives a time signal every hour
time=$(date +%I)  # what does this do? what is '+%I'?
count=0
# understand the following loop logic
while test $count -lt $time; do  # what is '-lt'
        echo -e "\a"  # what is '-e' for? what is "\a" for?
        sleep 1
        count=$[$count+1]
done
```

run `./time-signal.sh`

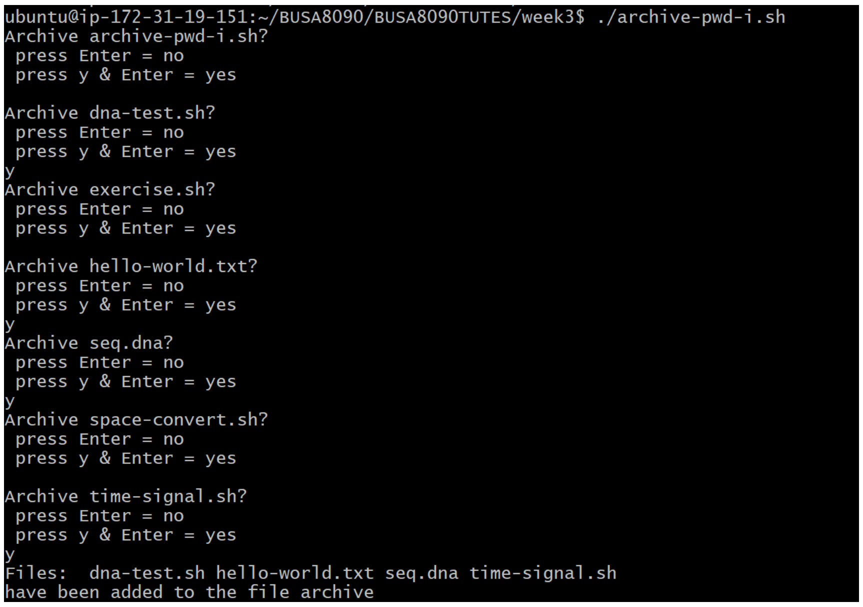
You should hear beeps for current hours

Select Files to Archive

```
#!/bin/bash
# save as archive-pwd-i.sh
# interactively archive files with tar
array=($(ls))
count=0
# understand the while loop
while test $count -lt ${#array[*]}; do  # what is '${#array[*]}'?
        echo "Archive ${array[count]}?"  # what is '${array[count]}'?
        echo " press Enter = no"
        echo " press y & Enter = yes"
        read input  # what does 'read' do?
        case $input in  # understand 'case in' control
```

```
                y*) list="${list} ${array[count]}"  # what is 'y*)'?
what does this line do?
        esac
        count=$(expr $count + 1)  # understand 'expr'
done
echo "Files: $list"
echo "have been added to the file archive"
tar -cf archive $list  # understand 'tar' command. what does '-cf' do?
```

run `./archive-pwd-i.sh`



```
ubuntu@ip-172-31-19-151:~/BUSA8090/BUSA8090TUTES/week3$ ./archive-pwd-i.sh
Archive archive-pwd-i.sh?
 press Enter = no
 press y & Enter = yes

Archive dna-test.sh?
 press Enter = no
 press y & Enter = yes
y
Archive exercise.sh?
 press Enter = no
 press y & Enter = yes

Archive hello-world.txt?
 press Enter = no
 press y & Enter = yes
y
Archive seq.dna?
 press Enter = no
 press y & Enter = yes
y
Archive space-convert.sh?
 press Enter = no
 press y & Enter = yes

Archive time-signal.sh?
 press Enter = no
 press y & Enter = yes
y
Files:  dna-test.sh hello-world.txt seq.dna time-signal.sh
have been added to the file archive
```

- ## Remove Spaces

```
#!/bin/sh
# save as space-convert.sh
chgname() {  # what is 'chgname()'
   echo "$1" | sed -e 's/[ ][ ]*/ /g' -e 's/[ ]/_/g'
   # understand 'sed' command
   # what do 's/' and '/g' mean?
   # what is this line doing?
}
find . -name '* *' | sort | while read name; do
file=$(basename "$name")
   stem=`dirname "$name"`  # `dirname "$name"` is equivalent to
$(dirname "$name")
   nfile=`chgname "$file"`
   nstem=`chgname "$stem"`
   if [ "$file" != "$nfile" ]
   then
      mv "$stem/$file" $nstem/$nfile
   fi
done
```

run `./space-convert.sh`

create a file with spaces in the name and then run this, see what happens