

# Module 1 Introduction to Version Control

## 1. Before Version Control

### (1) Goal of VCS

keep track of changes made to our files

### (2) Diffing file

```
diff rearrange1.py rearrange2.py
```

**Function:** compare the codes where difference lies in

- a. < means the corresponding line is removed from the 1st file
- b. means the corresponding line is added to the 2nd file
- c. If more than 1 section is changed:
  - (a) {number1}a{number2}: number 1 and 2 mean the lines in file 1 and file 2, a means added
  - (b) {number1}c{number2}: c means the line(s) at number 1 are changed to number 2 from file 1 to file 2

```
diff -u rearrange1.py rearrange2.py
```

**Function:** shows the change lines with some context

- a. - means line removed from 1st file
- b. + means line added in 2nd file
- c. Lines with No sign are context

#### **Additional tools for diffing files**

- a. wdiff function: highlights a word changed in a file instead of working line by line like diff does
- b. Graphical tools: meld, KDiff3, vimdiff.

### (3) Applying changes

```
diff -u old_file new_file > change.diff
```

**Function:** save the difference into change.diff file

- a. > in the code means to redirect the change to a new file “change.diff”
- b. Diff/Patch file: reflects all changes between two files

```
patch original_file < diff_file
```

**Function:** import the changes made in diff\_file into the original file

a. < in the code means to redirect the content of the file to standard input

- Reason why using diff/patch: The original code might be changed; use diff can avoid version update effects.

## (4) Useful codes in Bash collected from real-world example:

```
cp filename1 filename2
```

**Function:** copy file with filename1 and rename the copied file to filename2

```
./ filename
```

**Function:** run the code 'filename'

```
sys.exit({value})
```

**Function:** return the value of our script when it finishes (replace using return)

# 2. Version Control Systems (VCS)

## (1) Version control

- **commit:** make edits to multiple files and treat that collection of edits as a single change
- **use of VCS:**
  - stores the code, configuration, all essential document and their history
- know when the changes are made and who made the changes
  - revert a change if not a good idea or if an error comes up before solving it
- allow the author of a *commit* to record why the change was made (including bugs or issues fixed)

## (2) Git

- **Architecture:** distributed (*all people contributing on a repository have that repository on their own machines*)
- **Role:** Versatile (*work individually, act as server, act as client*)
- **Communication:** HTTP, SSH, Git's protocol
- **Official website:** [git-scm.com](https://git-scm.com) (scm stands for *source control management*)

## (3) Installation of Git

- **Git for windows:** <https://git-scm.com/downloads>

# 3. Using Git

## (1) First steps with git

- **Basic configurations:**

H3

```
git config --global user.email "me@example.com"
git config --global user.name "my name"
```

**Function:** set up the email and username for the git

```
git init
```

**Function:** choose a desired file folder for storage (cd xx) and set up an empty Git repository

```
ls -la
ls -l .git
```

**Function:** check the files in the folder (-la) and the subfolder .git (-l .git) directory

```
cp X:/xx.py .
ls -l
```

**Function:** copy a previous file to the working tree

```
git add xx.py
```

**Function:** make *Git* track our profile

```
git status
```

**Function:** Check the current status of *Git*

- Get the new file committed in *Git*

```
git commit
```

**Function:** Get the new file committed in *Git*

- **Note:** A commit message must be provided, or the 'commit' command will be aborted

- **Working tree:** the area out of a Git directory, works as a sandbox where we can edit the current version of the files

- **.Git directory:** acts as a database for all the changes and their history tracked in **Git**
- **Staging area (index):** a file maintained by *Git* that contains all information about files/changes into next commit

## (2) Tracking files

- **File type and stages:**
  - Track files (modified, -> staged, -> committed)
  - Untrack files (new files)

- **Actions after modifying a file:**

```
git add xx.py
```

**Function:** add the file to staging area

```
git commit -m 'comment'
```

**Function:** commit the file in the staging area with a comment

## (3) The basic Git Workflow

```
git config -l # check the configuration of the Git
git status # check the status: modified in red (unstaged)
git add xx.py # add a file to staging area
git status # check the status: modified in green (staged)
git commit # commit the changes
git status # working area is clean: no staged files
```

## (4) Commit Message

- **Good commit message: is broken up into a few sections:**
  - **First line:** short summary of the commit (< 50 characters)
  - **Second line:** blank line
  - **Next section:** full description of the necessary changes and interesting points (more paragraphs if needed), each line <72 characters
- **Actions after writing a commit message:**

```
git log
```

**Function:** check the previously written commit messages