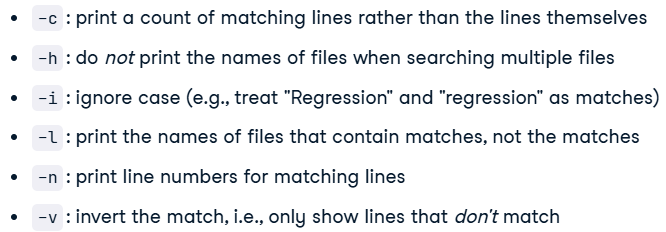
## Introduction to Shell

### Manipulating files and directories

* pwd – prints the *absolute path* of your **current working directory**;
* ls – lists the contents of your current directory;
  + ls /home/repl/seasonal
* A **relative path** specifies a location starting from where you are: it's like saying "20 kilometers north".
  + If it begins with /, it is absolute.
  + If it does not begin with /, it is relative.
* cd – changes the directory;
* .. "the directory above the one I'm currently in";
* ~ (the tilde character), which means "your home directory";
  + ls ~ will always list the contents of your home directory;
  + and cd ~ will always take you home.
* cp – copy file.
  + cp seasonal/summer.csv backup/summer.bck
    - (backup copies to the current directory);
  + cp seasonal/spring.csv seasonal/summer.csv backup
* mv – moving files from one direcory to another.
  + moving to another directory: mv seasonal/spring.csv seasonal/summer.csv backup
  + renaming files/directories: mv winter.csv winter.csv.bck
* rm – remove file.
  + rm people/agarwal.txt
* rmdir – removing directory (only if its ampty).
  + rmdir people
* mkdir – creating a directory.
  + mkdir people
* /tmp – created for a temprorary storing.
  + /tmp is immediately below the root directory /, not below your home directory.

### Manipulating data

* cat – prints the contents of the file on the screen.
  + its name is short for "concatenate", meaning "to link things together", since it will print all the files whose names you give it, one after the other.
  + cat course.txt
* less – one page is displayed at a time.
  + If you give less the names of several files, you can type :n (colon and a lower-case 'n') to move to the next file, :p to go back to the previous one, or :q to quit.
  + less seasonal/sping.csv seasonal/summer.csv
* head – show the start of the file.
  + head people/agarwal.txt
  + number of lines to show: head -n 5 seasonal/winter.csv
* TAB – to autocomplete the name of the file.
* ls –R – This shows every file and directory in the current level, then everything in each sub-directory, and so on.
  + ls -R -F /home
* man (manual) – get help from a command.
  + man ls
* cut – selects columns from a file
  + which means "select columns 2 through 5 and columns 8, using comma as the separator". cut uses -f (meaning "fields") to specify columns and -d (meaning "delimiter") to specify the separator.
  + cut -f 2-5,8 -d , values.csv
* !command – calls the most recent command use.
  + !n – call the n-th command.
* history – outputs all the commands called.
* grep – selects all lines according to what they contain.
  + grep molar seasonal/autumn.csv
  + additional info:



### Combining tools

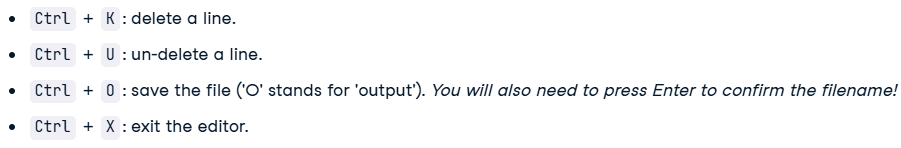
* > – redirecting the output to the file.
  + tail -n 5 seasonal/winter.csv > last.csv
* | – the pipe symbol tells the shell to use the output of the command on the left as the input to the command on the right.
  + cut -f 2 -d , seasonal/summer.csv | grep -v Tooth
* multiple selection using “|”:
  + cut -d , -f 2 seasonal/summer.csv | grep -v Tooth | head -n 1
* \* – to match zero or more values:
  + head -n 3 seasonal/s\*.csv
  + additional RE:
* ? matches a single character:
  + so 201?.txt will match 2017.txt or 2018.txt, but not 2017-01.txt.
* [...] matches any one of the characters inside the square brackets:
  + so 201[78].txt matches 2017.txt or 2018.txt, but not 2016.txt.
* {...} matches any of the comma-separated patterns inside the curly brackets:
  + so {\*.txt, \*.csv} matches any file whose name ends with .txt or .csv, but not files whose names end with .pdf.
* sort – sorts the output.
  + -n – sort numerically
  + –r – reverses the order of its output
  + -b –tells it to ignore leading blanks
  + -f – tells it be case-insensitive
  + cut -d , -f 2 seasonal/winter.csv | grep -v Tooth | sort -r
* uniq – outputs unique values (but have to be sorted).
  + cut -f 2 -d , seasonal/winter.csv | grep -v Tooth | sort | uniq -c
* ctrl-C – stops the programm.

### Batch processing

* echo + $ – printhing the variable value.
  + saving variable: testing=seasonal/winter.csv
  + applying to it: head -n 1 $testing
* looping: for filetype in docx odt pdf; do echo $filetype; done
* treating files with strange name: '2017 July data.csv'
* doing multiple things in a loop: for f in seasonal/\*.csv; do echo $f head -n 2 $f | tail -n 1; done

### Creating new tools

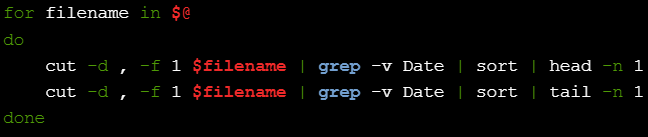
* nano filname – editing the file.
  + Info:



* recording the output: history > steps.txt
* bash filename – run commands in the file.
  + bash dates.sh
* @$ – accepts all of the command-line parameters given to the script (using code on your file, not on predefined one).
* predefining arguments: should select the second field from line 4 of seasonal/summer.csv.
  + func: head -n $2 $1 | tail -n 1 | cut -d , -f $3
  + creates: bash get-field.sh seasonal/summer.csv 4 2
* one script many things:
  + nano filename:



* looping through file contents:
  + example:

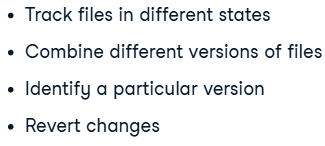


## Introduction to Git

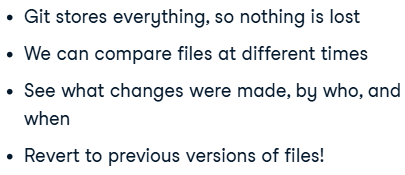
### Introduction to Git

#### Introduction to version control

* Version control – processes and systems to manage changes to files, programs and directories.
* What can version control do:

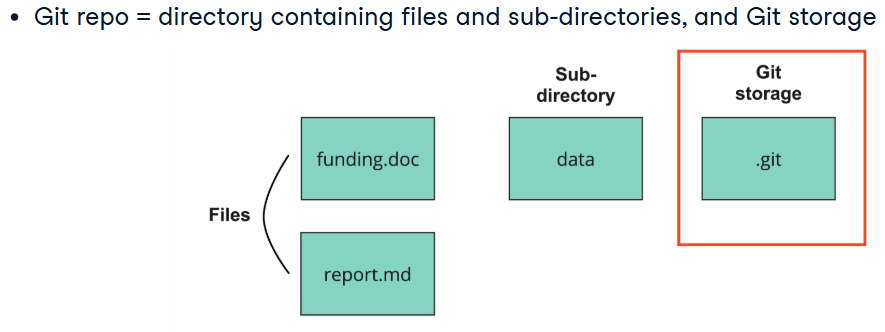


* Benefts of GIT:

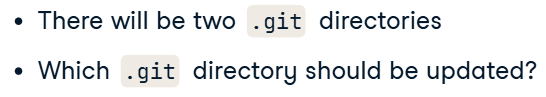


#### Creating repos

* GIT repo:

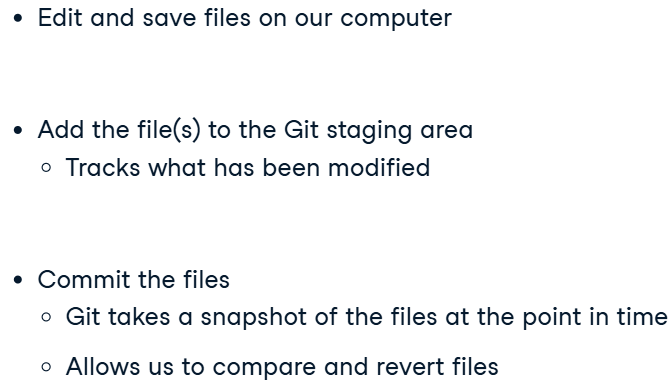


* + git init repo\_name – creating a repo.
  + git init (executed in a projects directory) – converts current project in the git repo.
* Why shouldn’t you to use subdirectories:



#### Staging and committing files

* GIT workflow:



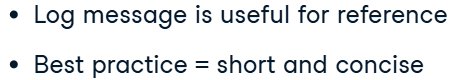
* Commits and staging:

We can think of adding files to the **staging area** as placing a letter in an envelope.

By contrast, **making a commit** is like putting the envelope in a mailbox.

We can add more things to the envelope or take things out as often as we want, but once we put it in the mailbox, we can't make further changes.

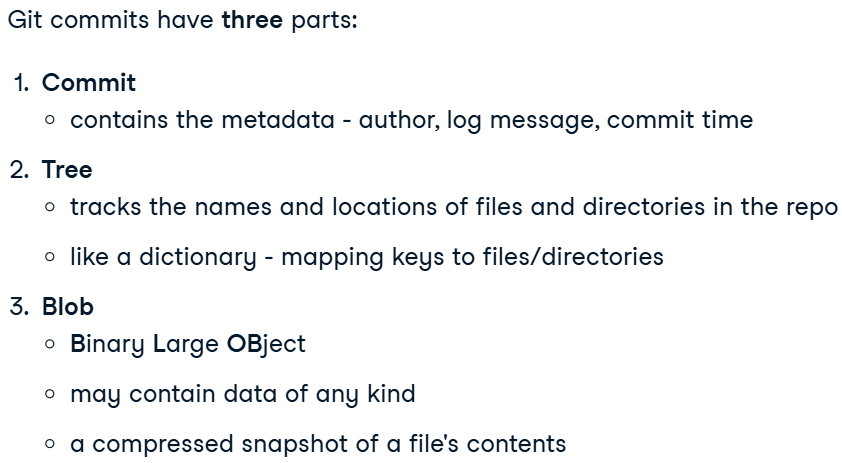
* git add file\_name (.) – add files to staging area.
* git commit –m “message” – commiting a file.
  + –m – allows a *log* massage without opening a text editor.
    - why are we using it:



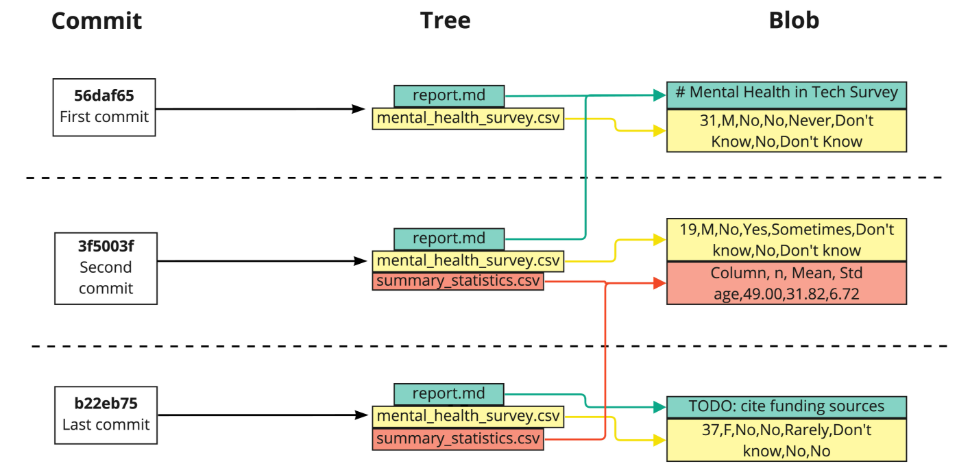
### Version history

#### Viewing the version history

* The commit structure:



* + How it works (how files with changes and without them are saved):



* git log – displaying all commits made to a repo in reverse chronological order (from recent to earliest one).
  + if there are a lot of commits – (:) will indicate more commits, so may press (Space) or q (to quit).

#### Version history tips and tricks

* git log –n – restricting output to a “n” files.
* git log file\_name – resrtict output to one file.
* Restricting git commit by a date:

 ‘Apr 21 2014’

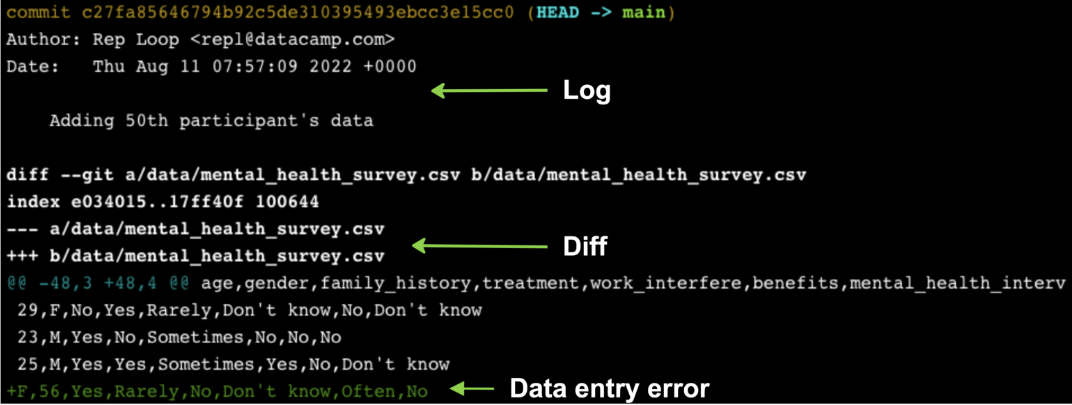
* + between dates:



* + What is better to use:



* git log output:

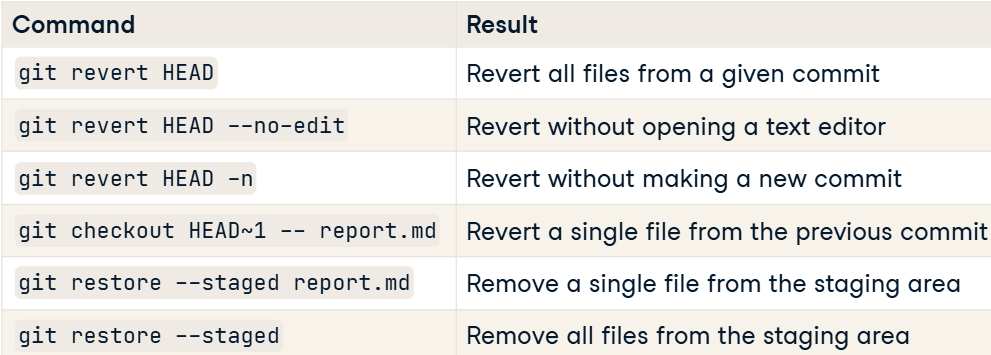


#### Comparing versions

* git diff – difference between commited versions.
* git diff --staged {file\_name}– difference between staged and commited version.
* git diff hash\_1 hash\_2
* git diff HEAD~1 HEAD – comparing the *second* most recent with the most *recent* commits.

#### Restoring and reverting files

* How to revert with staging and commiting:



## Intermediate Git

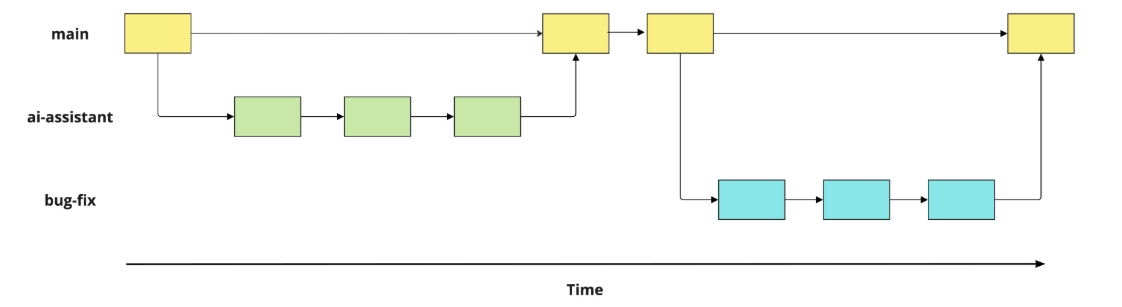
### Working with branches

#### Introduction to branches

* Branch – is an individual version of a repo.
  + Git uses it to systematically track multiple versions of files.
* Why use braches:



* Fixing bags by committing and merging branches:



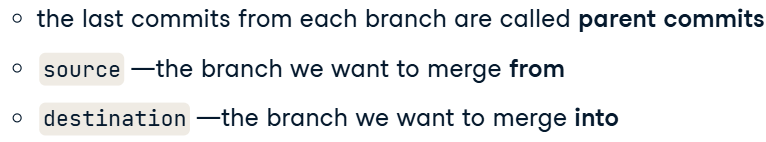
* git branch – lists branches of the project.
* git switch “branch” – moves to another branch.
* git branch “branch\_name” – creating a new branch.

#### Modifying and comparing branches

* git diff branch\_name main – difference between changed files and old ones.
* git branch –m old\_name new\_name – renaming a branch.
* git branch –d branch\_name – deleting a branch, after merging.

#### Merging branches

* Merging branches:

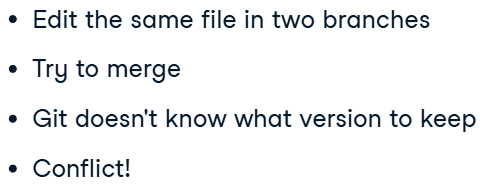


* git merge source\_b destination\_b – merging 1 into 2.

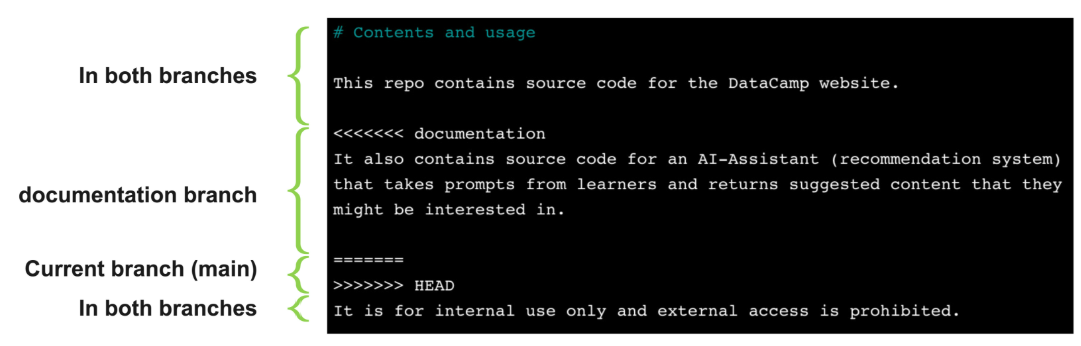
### Collaborating with Git

#### Merge conflicts

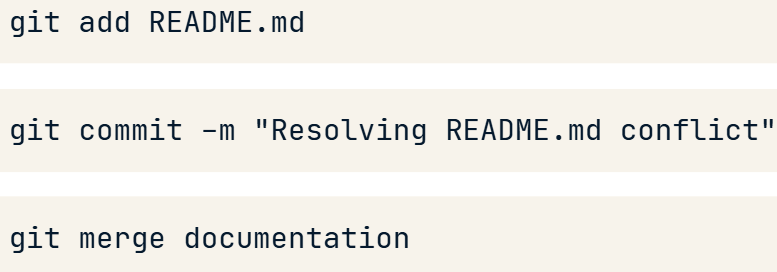
* Conflicts:



* Contents syntax:

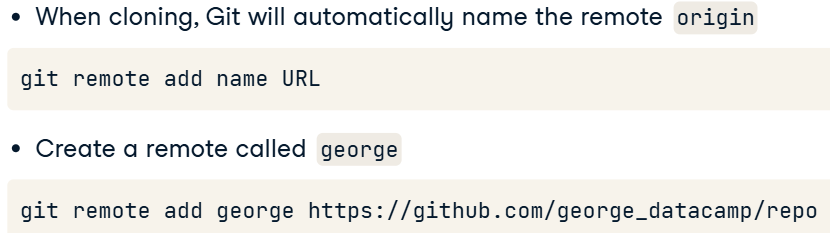


* nano file\_name – open and changing file contents.
* merging after resolving:



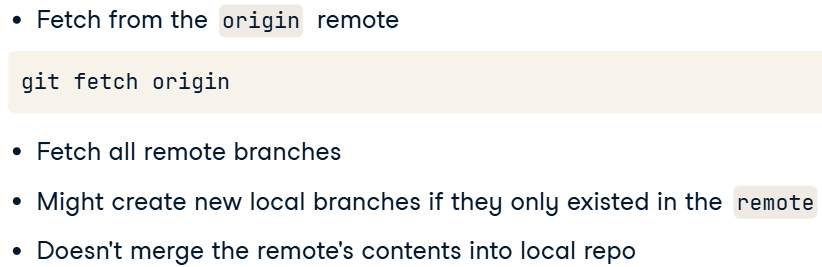
#### Introduction to remotes

* git clone path-to-project “new\_repo\_name”
* git clone URL – cloning from the remote repo.
* git remote – identifing a remote.
  + git remote –v – info about all remotes.
* **Когда ты клонируешь репозиторий, Git автоматически присваивает ему имя origin.**
* creating a remote:

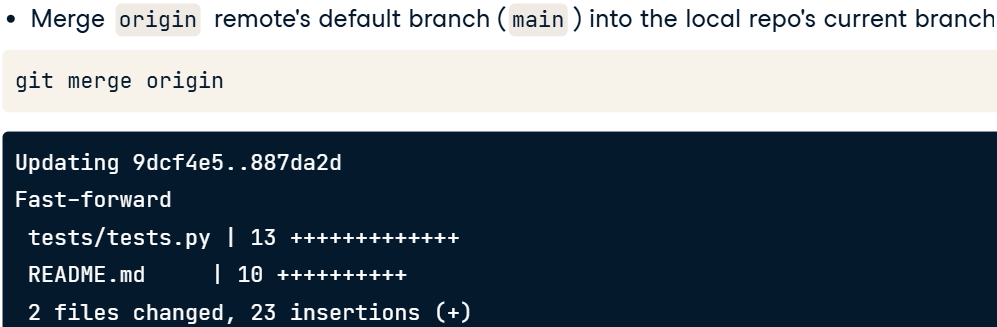


#### Pulling from remotes

* fetching from a repo:



* + git fetch origin main – fetching a specific (*main*) branch from the remote.
* synchronizing contents between local and remote repos:



* git pull origin origin\_branch – fetch and synchr. with “main” on the local (when we are in local branch).

#### Pushing to remotes

* before pushing – you should pull it.
* git push remote local\_branch – pushing into remote from local\_branch.
* Avoiding a conflict:
  + pull first from the remote:



* + commit changes locally;
  + push to a remote.