A Quick Intro to Version Control with git

https://github.com/gabindu/git-intro

Gabriel Indurskis, based on slides by Max Joseph

Nov. 12, 2025

Discuss

What is your current version control system?

1. How do you manage different file versions?

Discuss

What is your current version control system?

1. How do you manage different file versions?
2. How do you work with collaborators on the same files?

Discuss

What is your current version control system?

1. How do you manage different file versions?
2. How do you work with collaborators on the same files?
3. How much would your science/work/life suffer if your computer exploded right now? (scale from 1-10)

What is git

Version control system (VCS)

- manage different versions of files

What is git

Version control system (VCS)

- manage different versions of files
- collaborate with yourself

What is git

Version control system (VCS)

- manage different versions of files
- collaborate with yourself
- collaborate with other people

What is git

Version control system (VCS)

- manage different versions of files
- collaborate with yourself
- collaborate with other people
- in principle a commandline tool, but can be used conveniently via graphical interfaces (in particular in VS Code)

What is git

Version control system (VCS)

- manage different versions of files
- collaborate with yourself
- collaborate with other people
- in principle a commandline tool, but can be used conveniently via graphical interfaces (in particular in VS Code)
- various websites offer free remote storage (GitHub/GitLab/BitBucket)

Why use git

"Always remember your first collaborator is your future self, and your past self doesn't answer emails" - Christie Bahlai

Why use git

"Always remember your first collaborator is your future self, and your past self doesn't answer emails" - Christie Bahlai

- backup

Why use git

"Always remember your first collaborator is your future self, and your past self doesn't answer emails" - Christie Bahlai

- backup
- reproducibility

Why use git

"Always remember your first collaborator is your future self, and your past self doesn't answer emails" - Christie Bahlai

- backup
- reproducibility
- collaboration

Why use git

"Always remember your first collaborator is your future self, and your past self doesn't answer emails" - Christie Bahlai

- backup
- reproducibility
- collaboration
- organization

Why use git

"Always remember your first collaborator is your future self, and your past self doesn't answer emails" - Christie Bahlai

- backup
- reproducibility
- collaboration
- organization
- transparency

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:
  - usually nothing to do!

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:
  - usually nothing to do!
  - if necessary, `apt-get install git ssh`

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:
  - usually nothing to do!
  - if necessary, `apt-get install git ssh`
  - use your favourite editor (e.g. Emacs)

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:
  - usually nothing to do!
  - if necessary, `apt-get install git ssh`
  - use your favourite editor (e.g. Emacs)
  - use git on the commandline

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:
  - usually nothing to do!
  - if necessary, `apt-get install git ssh`
  - use your favourite editor (e.g. Emacs)
  - use git on the commandline
  - GUI alternatives:

Installation

- **On Mac OS**, git and ssh should already be available on the commandline. If not, install with Homebrew (install from https://brew.sh), using `brew install git ssh`
- **On Windows:** Install with a commandline package manager:
  - using `scoop` (needs to be installed first, from https://scoop.sh): `scoop install git ssh`
  - using `winget` (usually already included in Win 11): `winget install git ssh`
- **On Linux**:
  - usually nothing to do!
  - if necessary, `apt-get install git ssh`
  - use your favourite editor (e.g. Emacs)
  - use git on the commandline
  - GUI alternatives:
    - if you use Emacs, install `magit` package.

Initial Git & SSH configuration

- Set your name and email in Git:

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

```
git config --global user.name "Vlad Dracula"
git config --global user.email "vlad@tran.sylvan.ia"
git config --list
```

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

```
git config --global user.name "Vlad Dracula"
git config --global user.email "vlad@tran.sylvan.ia"
git config --list
```

- Create yourself an SSH key pair:

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

    ```
    git config --global user.name "Vlad Dracula"
    git config --global user.email "vlad@tran.sylvan.ia"
    git config --list
    ```
- Create yourself an SSH key pair:
  - In a terminal, do: `ssh-keygen -t ed25519`

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

    ```
    git config --global user.name "Vlad Dracula"
    git config --global user.email "vlad@tran.sylvan.ia"
    git config --list
    ```
- Create yourself an SSH key pair:
  - In a terminal, do: `ssh-keygen -t ed25519`
- Upload your public SSH key to GitHub (or GitLab or similar)

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

    ```
    git config --global user.name "Vlad Dracula"
    git config --global user.email "vlad@tran.sylvan.ia"
    git config --list
    ```
- Create yourself an SSH key pair:
  - In a terminal, do: `ssh-keygen -t ed25519`
- Upload your public SSH key to GitHub (or GitLab or similar)
  - Create yourself a free account

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

    ```
    git config --global user.name "Vlad Dracula"
    git config --global user.email "vlad@tran.sylvan.ia"
    git config --list
    ```
- Create yourself an SSH key pair:
  - In a terminal, do: `ssh-keygen -t ed25519`
- Upload your public SSH key to GitHub (or GitLab or similar)
  - Create yourself a free account
  - After logging on the website, click on your profile image, User Settings, SSH Keys

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

    ```
    git config --global user.name "Vlad Dracula"
    git config --global user.email "vlad@tran.sylvan.ia"
    git config --list
    ```
- Create yourself an SSH key pair:
  - In a terminal, do: `ssh-keygen -t ed25519`
- Upload your public SSH key to GitHub (or GitLab or similar)
  - Create yourself a free account
  - After logging on the website, click on your profile image, User Settings, SSH Keys
    - Copy & Paste your **public** key, usually found in `~/.ssh/id_ed25519.pub` (or maybe `id_rsa.pub` if you already had an older key)

Initial Git & SSH configuration

- Set your name and email in Git:
  - In a terminal:

    ```
    git config --global user.name "Vlad Dracula"
    git config --global user.email "vlad@tran.sylvan.ia"
    git config --list
    ```
- Create yourself an SSH key pair:
  - In a terminal, do: `ssh-keygen -t ed25519`
- Upload your public SSH key to GitHub (or GitLab or similar)
  - Create yourself a free account
  - After logging on the website, click on your profile image, User Settings, SSH Keys
    - Copy & Paste your **public** key, usually found in `~/.ssh/id_ed25519.pub` (or maybe `id_rsa.pub` if you already had an older key)
  - This enables a quicker way to up- and download files directly from the commandline, without the need of entering passwords.

Command line git

It's best to play around with git on the commandline at first, to better understand what it does. (Then it's ok to switch to a GUI.)

Command line git

It's best to play around with git on the commandline at first, to better understand what it does. (Then it's ok to switch to a GUI.)

Somewhere on your computer, create a new directory with a (text) file, e.g. `test.tex` or `test.txt`, and fill it with some example content (at least a few lines). (You could also just copy an already existing document.)

You can do this with your usual methods, or on the commandline, for example with:

```
mkdir my-first-git-repo
cd my-first-git-repo
echo "This is a fancy test!" > test.txt
```

You can also create other files, of whatever type you want (LaTeX, Markdown, HTML, Python scripts, ...) - binary files are ok as well!

Tell git to keep track of your files

Initializing a repository (only once)

On the commandline, make sure that you are inside the directory you created, then execute:

```
git init
```

If you call `ls -a`, you should now notice that a hidden `.git/` directory was created. This is where git does its magic, and you should therefore never touch this directory or its contents!

Tell git to keep track of your files

Initializing a repository (only once)

On the commandline, make sure that you are inside the directory you created, then execute:

```
git init
```

If you call `ls -a`, you should now notice that a hidden `.git/` directory was created. This is where git does its magic, and you should therefore never touch this directory or its contents!

Checking repository status

```
git status
```

You should notice that there are "untracked" files. Right now, git does not actually do anything with your files yet, we first have to tell it to "track" them.

Tell git to keep track of your files

Initializing a repository (only once)

On the commandline, make sure that you are inside the directory you created, then execute:

```
git init
```

If you call `ls -a`, you should now notice that a hidden `.git/` directory was created. This is where git does its magic, and you should therefore never touch this directory or its contents!

Checking repository status

```
git status
```

You should notice that there are "untracked" files. Right now, git does not actually do anything with your files yet, we first have to tell it to "track" them.

Adding your file to be tracked by git

To tell git that you want changes to the file `test.txt` to be "tracked", execute:

```
git add test.txt
```

Adding your file to be tracked by git

To tell git that you want changes to the file `test.txt` to be "tracked", execute:

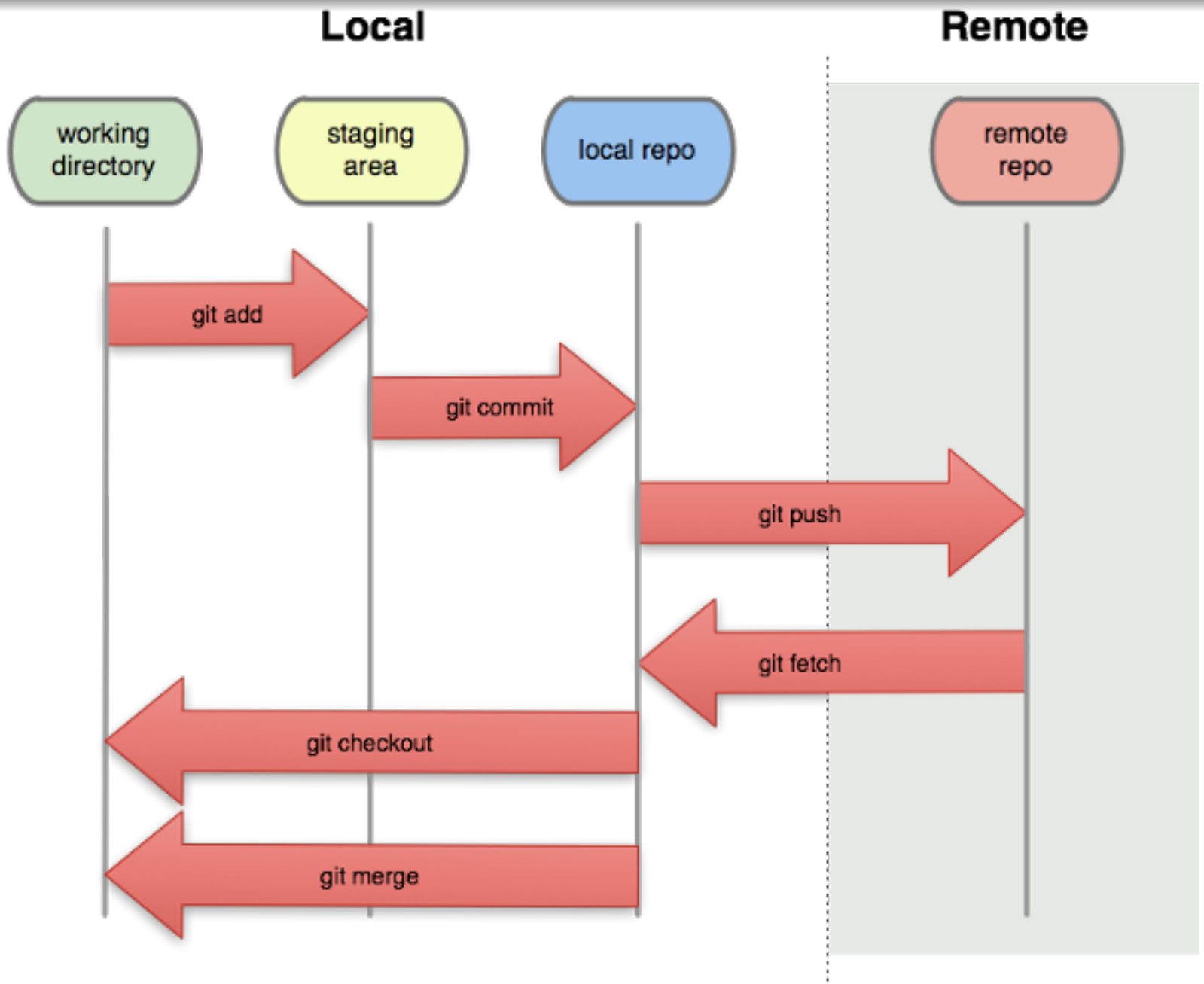
```
git add test.txt
```

or, to add all changed/new files (careful, this might add undesired temporary files):

```
git add --all
```

For future reference: If you want to avoid adding temporary files (like LaTeX auxiliary files etc.), you can add a file `.gitignore` which tells git to *never* even propose to track these files. You can download an appropriate `.gitignore` file for whatever type of document you're working on at https://www.gitignore.io/

Your changes are now "staged"



Committing

Changes aren't final until they're committed

```
git status
```

Committing

Changes aren't final until they're committed

```
git status
```

Committing

Once you're sure that your changes are worth saving
(THIS WILL GO ON YOUR PERMANENT RECORD)

```
git commit -m 'changed x, y, and z'
```

Committing

Changes aren't final until they're committed

```
git status
```

Committing

Once you're sure that your changes are worth saving
(THIS WILL GO ON YOUR PERMANENT RECORD)

```
git commit -m 'changed x, y, and z'
```

If you just use `git commit`, git will open an editor to ask you for a commit message. You can set the default editor by one of the following commands:

```
git config --global core.editor "atom --wait"
git config --global core.editor "emacs -nw"
git config --global core.editor "zile"
```

Commit messages

- Describe why and the what “in a nutshell”

Commit messages

- Describe why and the what "in a nutshell"
- Note to your future self (and to anyone else who you're collaborating with)

Commit messages

- Describe why and the what "in a nutshell"
- Note to your future self (and to anyone else who you're collaborating with)

Commit messages

- Describe why and the what "in a nutshell"
- Note to your future self (and to anyone else who you're collaborating with)

Now make more changes and repeat!

1. Change/add files (using whatever editor you prefer) - for your first test, change at least 2 different lines.

Now make more changes and repeat!

1. Change/add files (using whatever editor you prefer) - for your first test, change at least 2 different lines.
2. See a quick overview of what changed, using one or all of the following:

```
git status
git diff
git diff file
```

Now make more changes and repeat!

1. Change/add files (using whatever editor you prefer) - for your first test, change at least 2 different lines.
2. See a quick overview of what changed, using one or all of the following:

```
git status
git diff
git diff file
```

3. Add ("stage") changes with `git add file(s)`

Now make more changes and repeat!

1. Change/add files (using whatever editor you prefer) - for your first test, change at least 2 different lines.
2. See a quick overview of what changed, using one or all of the following:

```
git status
git diff
git diff file
```

3. Add ("stage") changes with `git add file(s)`
4. Commit changes with `git commit -m commit-message`

Now make more changes and repeat!

1. Change/add files (using whatever editor you prefer) - for your first test, change at least 2 different lines.
2. See a quick overview of what changed, using one or all of the following:

```
git status
git diff
git diff file
```

3. Add ("stage") changes with `git add file(s)`
4. Commit changes with `git commit -m commit-message`
5. View updated log with `git log`

Now, do something really stupid

- "Accidentally" introduce some errors to your file (or even delete a file!)

Now, do something really stupid

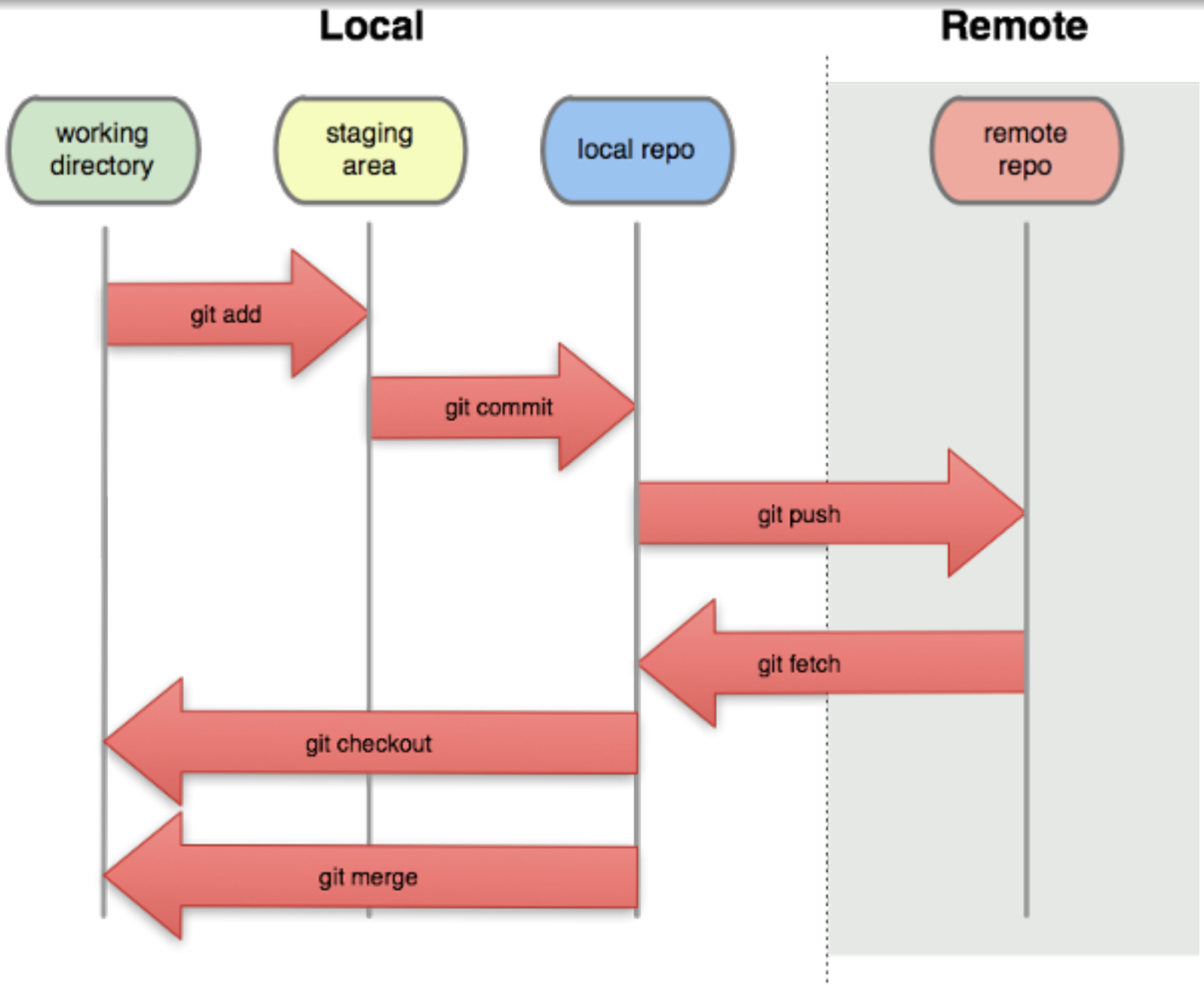
- "Accidentally" introduce some errors to your file (or even delete a file!)
- Whoops! *If only we had access to a time machine. . .*

Now, do something really stupid

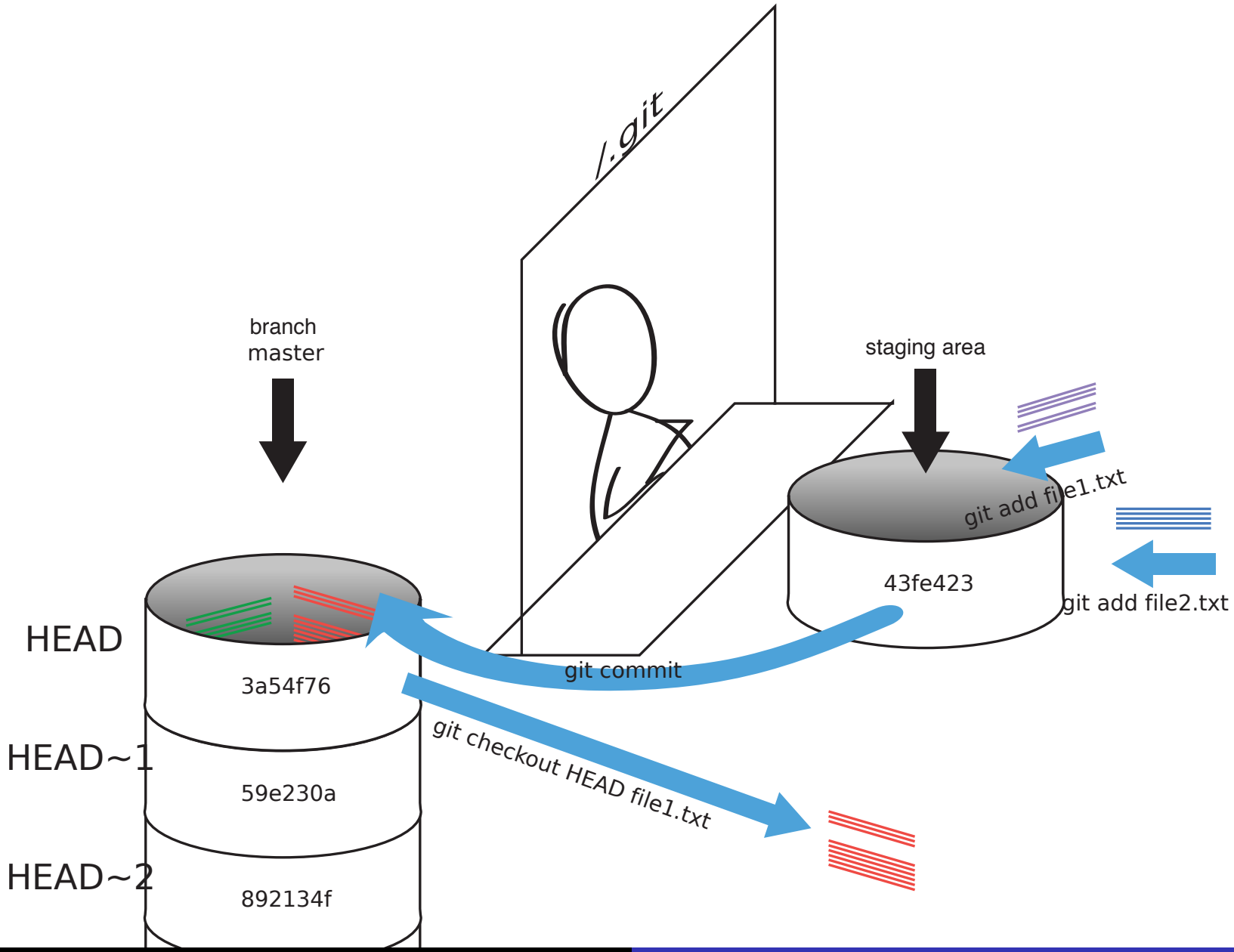
- "Accidentally" introduce some errors to your file (or even delete a file!)
- Whoops! *If only we had access to a time machine. . .*
- Hang on, we do!

```
git diff
git checkout HEAD test.txt
```

What happened?



Wait, what does HEAD refer to?



Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.
- free on BitBucket (w/ < 6 collaborators)

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.
- free on BitBucket (w/ < 6 collaborators)
- free on GitLab (**unlimited** collaborators)

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.
- free on BitBucket (w/ < 6 collaborators)
- free on GitLab (**unlimited** collaborators)
- all very similar, but differences include:

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.
- free on BitBucket (w/ < 6 collaborators)
- free on GitLab (**unlimited** collaborators)
- all very similar, but differences include:
  - feature set included in free vs. paid plan

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.
- free on BitBucket (w/ < 6 collaborators)
- free on GitLab (**unlimited** collaborators)
- all very similar, but differences include:
  - feature set included in free vs. paid plan
  - open source vs. closed source

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Up until now, everything has happened solely on your computer (and in fact, only in the directory you worked in). To have a backup and to synchronize between different computers (and possibly collaborators), you should link your repository to a "remote repository". There are several popular websites for this:

GitHub vs. GitLab vs. BitBucket

Private repos (only accessible by yourself or others you share it with):

- (only very recently) free on GitHub, but only < 4 collaborators.
- free on BitBucket (w/ < 6 collaborators)
- free on GitLab (**unlimited** collaborators)
- all very similar, but differences include:
  - feature set included in free vs. paid plan
  - open source vs. closed source
  - popularity & user base

I personally find GitLab the best free offer at the moment - but you can use all three if you want (and you can always switch)!

Mirroring your repository on the internet

Setting up a "remote"

1. Create an empty repository (or "project") on the GitLab/GitHub/BitBucket website (for now: no .gitignore, no README, and no license)

Mirroring your repository on the internet

Setting up a "remote"

1. Create an empty repository (or "project") on the GitLab/GitHub/BitBucket website (for now: no .gitignore, no README, and no license)
2. The website should show you instructions on what to do next, but if you already have the files and a git repository on your computer, it is simply:

   ```
   git remote add origin URL
   ```

   (use the URL shown on the website for your project, best the one using SSH, to avoid having to type in passwords all the time.)

Mirroring your repository on the internet

Setting up a "remote"

1. Create an empty repository (or "project") on the GitLab/GitHub/BitBucket website (for now: no .gitignore, no README, and no license)
2. The website should show you instructions on what to do next, but if you already have the files and a git repository on your computer, it is simply:

   ```
   git remote add origin URL
   ```

   (use the URL shown on the website for your project, best the one using SSH, to avoid having to type in passwords all the time.)
3. Verify the path of the remote:

   ```
   git remote -v
   ```

Synchronizing with the remote

Once your repository has been linked to a remote, you can:

Push (or "publish") your changes:

```
git push -u origin main
```

(after the first time, you can simply use `git push`)

Synchronizing with the remote

Once your repository has been linked to a remote, you can:

Push (or "publish") your changes:

```
git push -u origin main
```

(after the first time, you can simply use `git push`)

You can then check the remote website to see new changes. (Click on "Repository -> commits".)

Pulling from the remote

To get the latest changes (made by yourself or possibly collaborators) from the remote, do:

```
git pull
```

Synchronizing with the remote

Once your repository has been linked to a remote, you can:

Push (or "publish") your changes:

```
git push -u origin main
```

(after the first time, you can simply use `git push`)

You can then check the remote website to see new changes. (Click on "Repository -> commits".)

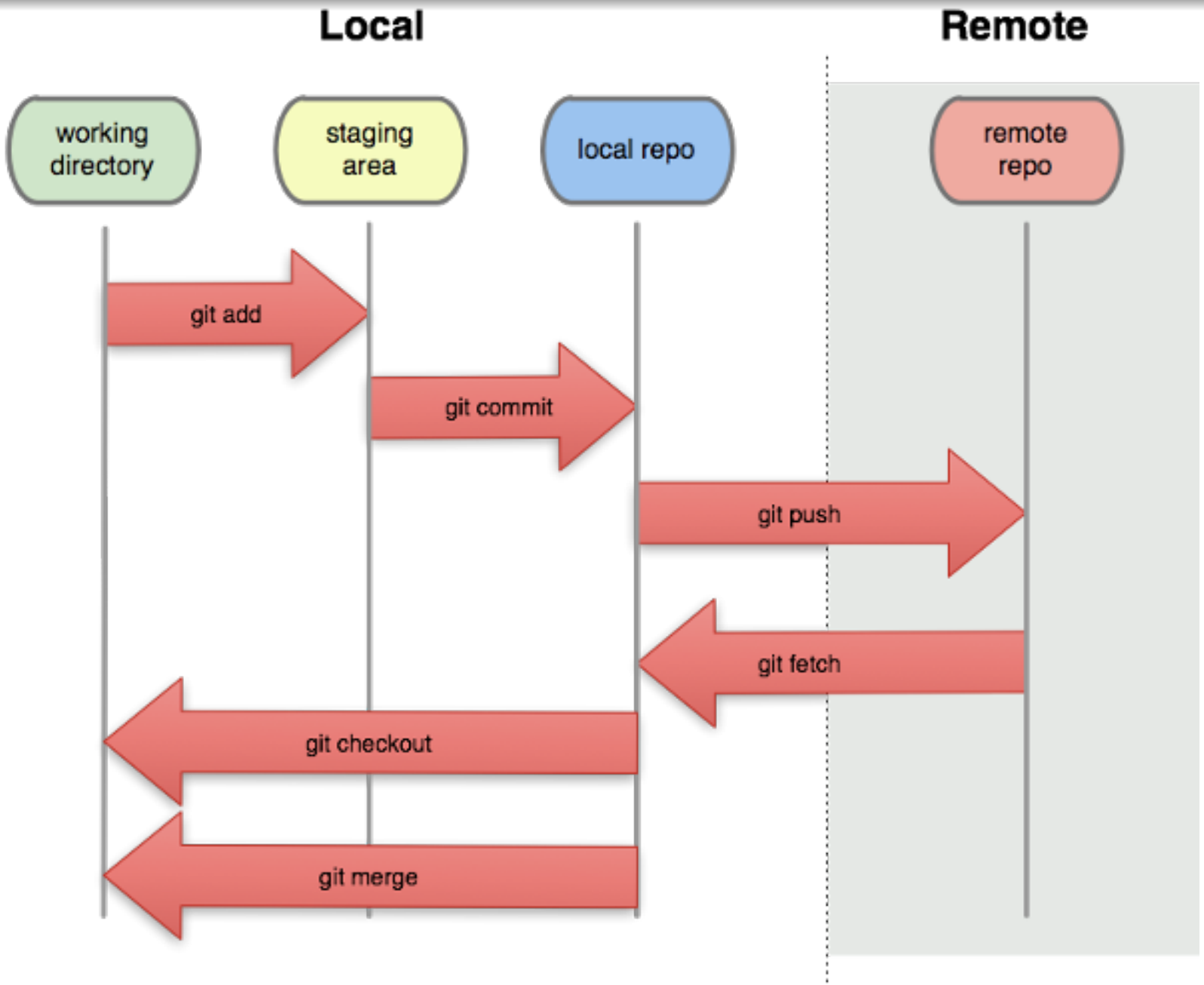
Pulling from the remote

To get the latest changes (made by yourself or possibly collaborators) from the remote, do:

```
git pull
```

Technical detail: `git fetch` only checks the status of the remote, while `git pull` actually applies those changes in your working directory.

Overview


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!
- Interface with other services (e.g. Slack.com) to get notifications on commits, discuss changes with team members...

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!
- Interface with other services (e.g. Slack.com) to get notifications on commits, discuss changes with team members...
- Collaborate with others!

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!
- Interface with other services (e.g. Slack.com) to get notifications on commits, discuss changes with team members...
- Collaborate with others!

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!
- Interface with other services (e.g. Slack.com) to get notifications on commits, discuss changes with team members...
- Collaborate with others!

Synchronize and continue work on a different computer

- If necessary, start from scratch by cloning your remote repo:

```
git clone URL
```

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!
- Interface with other services (e.g. Slack.com) to get notifications on commits, discuss changes with team members...
- Collaborate with others!

Synchronize and continue work on a different computer

- If necessary, start from scratch by cloning your remote repo:

  `git clone URL`
- Update the local repo from the remote with: `git pull`

Things you can do with a remote repository

Use the fancy website interface

- examine your code, the commit log, etc. online
- keep track of issues, things to-do, etc.
- you can even edit your files online, using (on GitHub) an online version of VS Code!
- Interface with other services (e.g. Slack.com) to get notifications on commits, discuss changes with team members...
- Collaborate with others!

Synchronize and continue work on a different computer

- If necessary, start from scratch by cloning your remote repo:

  `git clone URL`
- Update the local repo from the remote with: `git pull`
- Important rule to remember: Always `git pull` before starting to edit your local files!

“Forking” an already existing repository

If you want to base your work on something somebody else already has created, you can create a “fork” of their online repository. We’ll use a repository I created on Github as an example:

- Make sure you are logged into Github (create yourself a free account if necessary).

“Forking” an already existing repository

If you want to base your work on something somebody else already has created, you can create a “fork” of their online repository. We’ll use a repository I created on Github as an example:

- Make sure you are logged into Github (create yourself a free account if necessary).
- Find the URL of a repository you want to work on. For example, go to https://github.com/gabindu/git-playground

“Forking” an already existing repository

If you want to base your work on something somebody else already has created, you can create a “fork” of their online repository. We’ll use a repository I created on Github as an example:

- Make sure you are logged into Github (create yourself a free account if necessary).
- Find the URL of a repository you want to work on. For example, go to https://github.com/gabindu/git-playground
- Click on the small “Fork” button near the top right of the page. This creates a copy of this repository (including all its history), but under *your* account.

"Forking" an already existing repository

If you want to base your work on something somebody else already has created, you can create a "fork" of their online repository. We'll use a repository I created on Github as an example:

- Make sure you are logged into Github (create yourself a free account if necessary).
- Find the URL of a repository you want to work on. For example, go to https://github.com/gabindu/git-playground
- Click on the small "Fork" button near the top right of the page. This creates a copy of this repository (including all its history), but under *your* account.
- This new repository is yours to do with as you please, it is independent of the original source repository. (But it still remembers where it came from, more on that later.)

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline
  - In a terminal on your computer, do:

```
cd folder-where-you-want-it
git clone URL
```

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline
  - In a terminal on your computer, do:

    ```
    cd folder-where-you-want-it
    git clone URL
    ```

- Method 2: In Visual Studio Code

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline
  - In a terminal on your computer, do:

    ```
    cd folder-where-you-want-it
    git clone URL
    ```

- Method 2: In Visual Studio Code
  - Open a new window (if necessary)

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline
  - In a terminal on your computer, do:

    ```
    cd folder-where-you-want-it
    git clone URL
    ```

- Method 2: In Visual Studio Code
  - Open a new window (if necessary)
  - Click on "Clone Git Repository"

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline
  - In a terminal on your computer, do:

    ```
    cd folder-where-you-want-it
    git clone URL
    ```

- Method 2: In Visual Studio Code
  - Open a new window (if necessary)
  - Click on "Clone Git Repository"
  - Paste the URL you copied earlier.

Clone your remote repository

Now let's get the files onto your own computer:

- On the main page of *your* copy of the `git-playground` repository, click on the green "Code" button, then click the "copy" icon next to the shown URL.
- Method 1: manually on the commandline
  - In a terminal on your computer, do:

    ```
    cd folder-where-you-want-it
    git clone URL
    ```

- Method 2: In Visual Studio Code
  - Open a new window (if necessary)
  - Click on "Clone Git Repository"
  - Paste the URL you copied earlier.
- This automatically connects your new local repo with the remote, so you can directly use `git push` and `git pull`.

Contributing back to the original repository

- Once you have made some changes and you have pushed them to (your) remote repository, you can ask the original author to "pull in" your changes (if you think they are worthy to be shared!).

Contributing back to the original repository

- Once you have made some changes and you have pushed them to (your) remote repository, you can ask the original author to "pull in" your changes (if you think they are worthy to be shared!).
- Since GitHub remembers where your repository originally came from, it will show you (on the website) how it differs from the original source. For example, it might say that it is "8 commits ahead of . . . "

Contributing back to the original repository

- Once you have made some changes and you have pushed them to (your) remote repository, you can ask the original author to "pull in" your changes (if you think they are worthy to be shared!).
- Since GitHub remembers where your repository originally came from, it will show you (on the website) how it differs from the original source. For example, it might say that it is "8 commits ahead of ..."
- You can now create a so-called **Pull Request**: This sends a message to the original author, offering to "merge" your changes into their original repository.

Contributing back to the original repository

- Once you have made some changes and you have pushed them to (your) remote repository, you can ask the original author to "pull in" your changes (if you think they are worthy to be shared!).
- Since GitHub remembers where your repository originally came from, it will show you (on the website) how it differs from the original source. For example, it might say that it is "8 commits ahead of ..."
- You can now create a so-called **Pull Request**: This sends a message to the original author, offering to "merge" your changes into their original repository.
- This is the basic principle of how you can **contribute to an open-source project**!

Branches

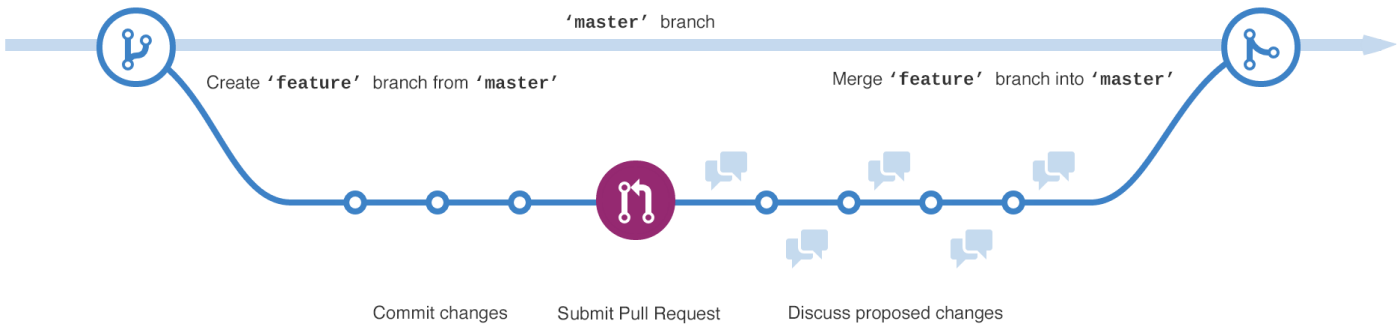
- Any repository has a default "branch" in which all files are stored, usually called "main". This branch is usually reserved for the current most up-to-date, well-working production version (good example to keep in mind: the live files for a website, e.g. http://math.mychamplain.ca)

Branches

- Any repository has a default "branch" in which all files are stored, usually called "main". This branch is usually reserved for the current most up-to-date, well-working production version (good example to keep in mind: the live files for a website, e.g. http://math.mychamplain.ca)
- But when working on new "features", it's usually not a good idea to immediately put those into the main branch!

Branches

- Any repository has a default "branch" in which all files are stored, usually called "main". This branch is usually reserved for the current most up-to-date, well-working production version (good example to keep in mind: the live files for a website, e.g. http://math.mychamplain.ca)
- But when working on new "features", it's usually not a good idea to immediately put those into the main branch!
- So, instead, you create a new branch (or fork!), work in there without danger of destroying anything for others, and finally ask for the changes to be **merged** into the main branch:



Working in a branch

Create a local branch

- Create & checkout a new branch: `git checkout -b branchname`

Working in a branch

Create a local branch

- Create & checkout a new branch: `git checkout -b branchname`
- Work on the files as before, stage, commit, and push to the remote server.

Working in a branch

Create a local branch

- Create & checkout a new branch: `git checkout -b branchname`
- Work on the files as before, stage, commit, and push to the remote server.
- Inspect the log to see what happened (`git log`)

Working in a branch

Create a local branch

- Create & checkout a new branch: `git checkout -b branchname`
- Work on the files as before, stage, commit, and push to the remote server.
- Inspect the log to see what happened (`git log`)

Working in a branch

Create a local branch

- Create & checkout a new branch: `git checkout -b branchname`
- Work on the files as before, stage, commit, and push to the remote server.
- Inspect the log to see what happened (`git log`)

Merge your changes into `main`

When you're satisfied with your work (and you pushed to the remote), it's time to "merge" it into the main branch. If you're the owner of both the original and the new branch, you can do this yourself, with the `git merge` command (or via the GUI in VSCode).

But if the original repository is owned by somebody else, you need to **create a "Pull Request"**, which is easiest done on the website, using "Contribute" - "Create Pull Request".

General Best Practice Rules of Thumb

- **Commit often, perfect later, publish once!**

General Best Practice Rules of Thumb

- **Commit often, perfect later, publish once!**
- Use meaningful commit messages.

General Best Practice Rules of Thumb

- **Commit often, perfect later, publish once!**
- Use meaningful commit messages.
- Do *not* commit anything that can be regenerated from other things that were committed. For example, pdf-files which are created from a LaTeX source, binary files which were compiled from a C++ source file, etc.

General Best Practice Rules of Thumb

- **Commit often, perfect later, publish once!**
- Use meaningful commit messages.
- Do *not* commit anything that can be regenerated from other things that were committed. For example, pdf-files which are created from a LaTeX source, binary files which were compiled from a C++ source file, etc.
- Note: The case of pdf-files is arguable - it's sometimes nice to have the latest compiled document version archived as well, but it is technically an unnecessary waste of harddrive space and causes the repository to grow much faster than it usually would. Use your own judgment.

General Best Practice Rules of Thumb

- **Commit often, perfect later, publish once!**
- Use meaningful commit messages.
- Do *not* commit anything that can be regenerated from other things that were committed. For example, pdf-files which are created from a LaTeX source, binary files which were compiled from a C++ source file, etc.
- Note: The case of pdf-files is arguable - it's sometimes nice to have the latest compiled document version archived as well, but it is technically an unnecessary waste of harddrive space and causes the repository to grow much faster than it usually would. Use your own judgment.
- *Always* use `git pull` before you start editing. This pulls in any changes made by others (or yourself on another computer!) from the remote repository.

Using a graphical user interface (GUI) to git

Now that you're comfortable with the principles behind git, you are ready to do everything with a few clicks (instead of typing `git add`, `git commit`, etc. all the time)!

In Visual Studio Code

VS Code has `git` built-in, and you can access it using the (usually) third button on the left.

- Changes are shown as a list of modified files, clicking on each shows you the difference ("diff") to the previous commit in a side-by-side view.

Using a graphical user interface (GUI) to git

Now that you're comfortable with the principles behind git, you are ready to do everything with a few clicks (instead of typing `git add`, `git commit`, etc. all the time)!

In Visual Studio Code

VS Code has `git` built-in, and you can access it using the (usually) third button on the left.

- Changes are shown as a list of modified files, clicking on each shows you the difference ("diff") to the previous commit in a side-by-side view.
- There is no need to stage changes, any modified (or new) file is automatically considered staged when you commit.

Using a graphical user interface (GUI) to git

Now that you're comfortable with the principles behind git, you are ready to do everything with a few clicks (instead of typing `git add`, `git commit`, etc. all the time)!

In Visual Studio Code

VS Code has `git` built-in, and you can access it using the (usually) third button on the left.

- Changes are shown as a list of modified files, clicking on each shows you the difference ("diff") to the previous commit in a side-by-side view.
- There is no need to stage changes, any modified (or new) file is automatically considered staged when you commit.
- To commit, you simply enter your commit message at the top, and press "Enter" (or the blue "Commit" button).

Using a graphical user interface (GUI) to git

Now that you're comfortable with the principles behind git, you are ready to do everything with a few clicks (instead of typing `git add`, `git commit`, etc. all the time)!

In Visual Studio Code

VS Code has `git` built-in, and you can access it using the (usually) third button on the left.

- Changes are shown as a list of modified files, clicking on each shows you the difference ("diff") to the previous commit in a side-by-side view.
- There is no need to stage changes, any modified (or new) file is automatically considered staged when you commit.
- To commit, you simply enter your commit message at the top, and press "Enter" (or the blue "Commit" button).
- The log of recent commits is automatically shown at the bottom.

Additional ressources

References

- Git it: Interactive Tutorial to learn some more details about git

Additional ressources

References

- Git it: Interactive Tutorial to learn some more details about git
- Pro Git: free book by Scott Chacon and Ben Straub with everything you might ever want to know about git

Additional ressources

References

- Git it: Interactive Tutorial to learn some more details about git
- Pro Git: free book by Scott Chacon and Ben Straub with everything you might ever want to know about git
- Git Cheat Sheet

Additional ressources

References

- Git it: Interactive Tutorial to learn some more details about git
- Pro Git: free book by Scott Chacon and Ben Straub with everything you might ever want to know about git
- Git Cheat Sheet
- On undoing, fixing, or removing commits in git: A git choose your own adventure

Additional ressources

References

- Git it: Interactive Tutorial to learn some more details about git
- Pro Git: free book by Scott Chacon and Ben Straub with everything you might ever want to know about git
- Git Cheat Sheet
- On undoing, fixing, or removing commits in git: A git choose your own adventure
- https://www.gitignore.io/: Create appropriate `.gitignore` files for your projects

Additional ressources

References

- Git it: Interactive Tutorial to learn some more details about git
- Pro Git: free book by Scott Chacon and Ben Straub with everything you might ever want to know about git
- Git Cheat Sheet
- On undoing, fixing, or removing commits in git: A git choose your own adventure
- https://www.gitignore.io/: Create appropriate `.gitignore` files for your projects
- Git Best Practices