start
here -->

download Git on your server or PC

- sudo yum install git
- sudo yum install git -y (does the same thing

except it allows you to skip the (yes or no) prompt. hence -y)

to install it on your windows system

- download gitbash from any browser

create a development directory for git.
Then initialise the directory

1. mkdir *directory*
2. cd *directory*
3. git init

add user

1. git --global user.email *email*
2. git --global user.name *name*

add repository

OPTION 1: (with this option, though it is quicker to set up, you will need to authenticate yourself every time you try connect to the server)

- git remote add *alias-name https-url*
-

OPTION 2:

- git remote add *alias-name ssh-url*
- ssh -keygen
- *copy the public key that it prints out*
- * if no public key is printed then: cd / --> cat ./.ssh/id_rsa   --> copy this public key
- *go to github --> settings --> "SSH and GPG keys" section --> click "new ssh key" --> make a name for your key --> paste your generated public key in the space for "key"

lastly. Some files (such as .classpath, .project etc..) will come with the coding files
We don't want to track those files
The only thing that we want to track are the coding files.
examples of coding files would be files with the following extensions:

- .py
- .java
- .delphi
- etc...

To ensure that these files do not get tracked we move them into a file called a ".gitignore" file. Git will ignore this file/ it will be hidden

1. touch .gitignore
2. mv .classpath .gitignore
3. repeat step 2 with any remaining files that you do not want to track

if you're confused about any command use

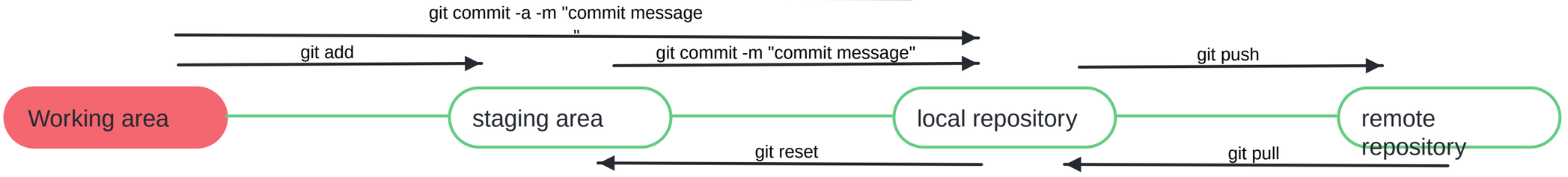
command -help

eg.
git branch -help

(btw this is only page 1. There are 3)

the button to switch pages is in the bottom right. It looks like a square with a number in it

"git commit" command:
• creates a current version of files
• file is now committed
• when you use this command you will need to be authenticated (asks for username and token)



• coding/developing done here.

• anytime you make changes to a file it comes back to this unsaved area. this is when it is recommended to use the **git commit -a -m command**. The "-a" means you can skip redoing the add step

• when you're ready to commit, use git add
• when git status command used, files in this area will appear red

• tracked files are seen here.

• once you're done with a task commit it.
• when git status command used, files in this area will appear green

• files here are in your local "version" of the remote repository

• only the user who made the commit can see these files & the changes made. In order for other team members to see, you need to push it to the
remote repository.

• if you want to delete a file you've just commited, you can use git revert command.
•

• all users with permissions have access now.

• file and changes made shows up as a version on github

There is more information about tracking files and areas on page 2

Congratulations!! You've set up your git environment successfully
Now it's time to create a file.
- touch *file-name*

After the file has been created and editted,
here are the possible tasks that you may need to run

THE FIRST THING YOU SHOULD DO IS,
TO START TRACKING YOUR FILE (git add)
unless you do, it will stay in the staging area
- git add file-name (to add everytging use "." in place of file-name)

to delete files from being tracked use
- git rm

to delete files form staging area use git clean
- git clean (deletes files in working area (untracked= red))

git clean -n (indicates which files will be deleted)
git clean -f (deletes those files)
git clean -f -d (delete whole directories)
git clean -f -x (remove ignored files along with everything else deleted)
git clean -f -X (delete only the ignored files)

once you're done editing the file to your liking/ you've finished the task
commit the file into your local directory

- git commit -a -m “messge”

if you change your mind and want to remove the file from your working
area you can send it back with..

- git reset commit-ID

to delete a file you've commited

- git revert commit-ID

to enhance collaboration use the git push command to send the file to the remote repository (if you have the token already you can skip steps 2-5)

1. git push
2. *go to GitHub -> developers settings
3. go to personal access token
4. generate taken
5. copy and save this somewhere secure
6. paste generated token where
a password is required

Note that you need to have relevant permissions to do both commands to fetch/pull a file from your remote repository into your local repo. Use...

- git pull alias-name branch (brings changes to local repo)
- git fetch alias-name branch (brings changes to working area)
- git clone https-url (beings entire code to working area, like copy and paste. You can only clone a public repo)

COMMANDS FOR BRANCHING
- git branch (see which branch you're in)
- git branch branch-name (create a branch)
- git diff branch-name (see difference between current branch and another)
- git switch branch-name (switch from current branch to another -- make sure work tree is clean = no files in working area/ everything added and committed/ or use git stash to back it up and then remove from your work tree)
- git checkout branch-name (switches from one branch to another but it commits everything for you? (i think))
- git merge (merge changes from downstream branch to upstream) (if there is a merge conflict error, vi into the file with the error ---> delete ">>" and "<<" and "==" lines where applicable ---> git commit -a -m "merge conflict resolved")
- git rebase (works the same as git merge except instead of storing commits as one lump commit it transfers them as individual commits)

git merge: commit 1-- commit 2--commit 3 merged = just one lump commit
git rebase: commit 1 -- commit 2 --commit 3 merged = commit 1 -- commit 2--commit 3

- git cherry-pick commit-ID (only merges specified commit with branch)
- git branch branch-name -d (delete branch)
- git push alias-name --all (push all branches to remote repo)

COMMANDS FOR TAGS
- git tag tag-name (create a tag)
- git tag -d tag-name (delete tag)
- git tag push alias-name tag tag-name (push tag to remote repo)
- git tag push alias-name --tags (push all tags to remote repo)

OTHER COMMANDS
- git stash (backs up everything in working area and then deletes it)
- git stash list (stashes everything listed in an index from 0 upwards eg. 0,1,2...)
- git stash apply stash @{x} (brings stash at index "x" (any number) into your working area)
- git stash apply (brings the most recent one stashed back into working area)
- git stash drop (deletes everything stashed)
- git stash drop stash@{x} (deletes stash at index "x")
- git stash pop (runs git apply and git drop to all things stashed)
- git stash pop stash@{x} (^ but for a the stash at index "x)

The difference between tags and branches:

Branches:
- mutable
- modifiable
- used for development (work in progress)
- you can create them from any branch (doesn't all have to come from the master branch)

Tags:
- incommutable
- unmodifiable
- used for after production
- it's recommended that you create them with master as the upstream

COMMANDS FOR TRACKING FILES AND CHANGES MADE ETC.

- git status (shows ^ red= untracked, green= tracked)
- git log (this shows what has been commited)
- git show (shows what has been commited + changes that were made for each commit)