Karlsruher Institut für Technologie

Git Manual: The Subject-Oriented Approach

Author: Anh Kha Nguyen
Semester: 7. Bachelor Semester
Supervisor: Dr.-Ing. Dipl.-Wi.-Ing. Matthes Elstermann

Winter Semester 2020

Institute for Information Management and Engineering
www.imi.kit.edu

KIT - The Research University in the Helmholtz Association

Contents

1 Introduction

This chapter gives a brief overview of the the document's overall motivation, its current state and the needed background knowledge that is required to read this document.

1.1 Motivation and current state

As the title suggests, the document's overall motivation is to deliver a subject-oriented approach of a git manual. Git is a Version Control System (VCS) developed by Linus Torvalds [LM12]. Compared to other VCSs, The main difference of git is that it has a different approach to file versioning and collaboration development, as both are much more efficient in git [CS20]. This makes git one of the most popular VCS out there, which is why the provision of git instruction is of particular interest. The end goal of this document is to provide the *ultimate* git manual. Its final version should aspire to provide all the necessary knowledge about git to any reader, regardless of his/her technical background.

As of the document's current state, it provides 12 git commands which lay a solid foundation for any basic git workflow. With the provided commands, the reader is able to:

- **create** a git repository (2.1)
- **add** (2.2) and **commit** (2.3) any file within a git repository
- **check the current state** of a git repository at any given time (2.4)
- **view** all existing branches and **create** new ones (3.1)
- **switch** to any existing branch (3.2)
- **merge** different branches of a git repository (3.3)
- **set a new branch beginning** for any branch of a git repository (3.4)
- **save** a copy of a remote git repository into the local storage (4.1)
- **fetch** commits of a remote repository (4.2)
- **pull** (fetch and apply) commits of a remote repository on any branch (4.3)
- **push** commits of any branch of a repository to its local counterpart (4.4)

This set of commands is enough for the majority of git workflows as the conjunction of all commands enables the usage of git's VCS functionality (2), as well as the basic functionalities of branching (3) and collaborations (4).

1.2 Foundations and related work

There are three concepts that are required to fully understand this manual. The first concept is that git has introduced three different states for a file residing in a git repository.

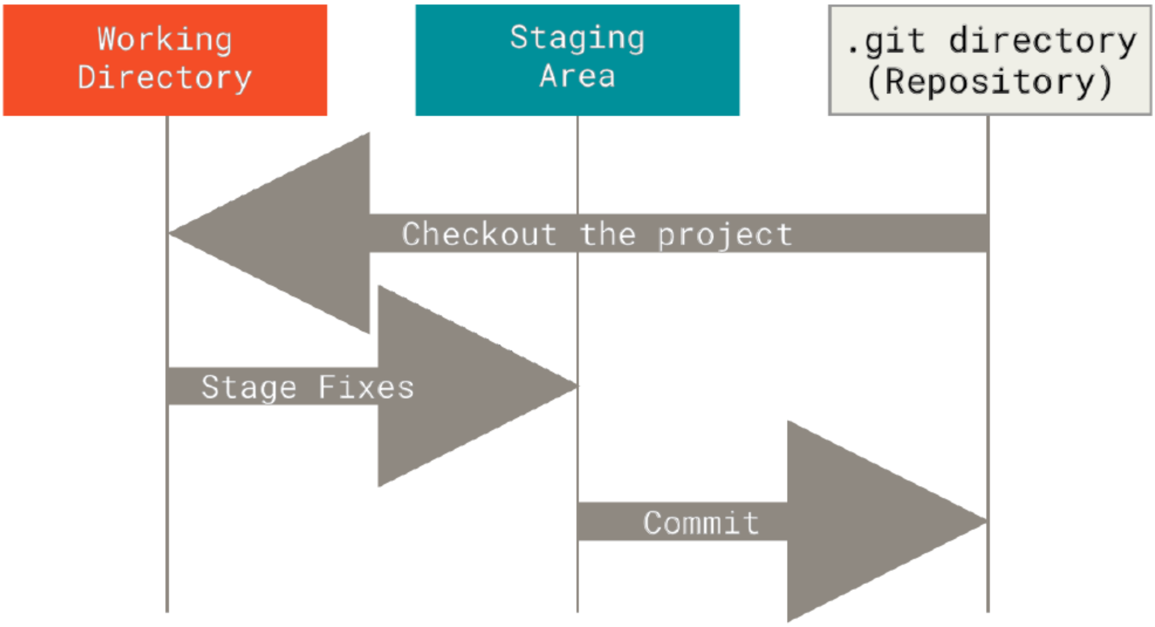


Figure 1.1: These are the transitions that happen in a state change[CS20]

The three states are: *modified*, *staged* and *committed*. If a file is modified, it means that the file has been changed or worked on, but is not committed yet. To do that, the file has to be staged first. This means that the modified file is marked to go into the next commit. The file which keeps track of the marked files is called the "index". If the user commits the staged file, the file is in its committed state which means that the data, containing the changes, is safely stored in the git repository

As seen in 1.1, there are three locations the files can be in during the work with a git repository: the *working directory*, *staging area* and *.git directory*. This is the second concept to remember during the read of the manual. By opening any file within a git repository, git will automatically checkout the latest version of the project by translating and putting the compressed files within the repository onto the local disk. These modified files will then go through the three states depending on the workflow.

The third concept is completely unrelated to git and revolves around the process modeling schema itself. In the scope of this document, the Parallel Activity Specification Schema (PASS) is used to give a subject-oriented perspective of what process are involved during the interaction of a git user and the git software. Both are used as the main subjects of all Subject Interaction Diagrams (SID). Each git command has at least one SID. More details about PASS and its properties are found here [El21]. Here are some detailed descriptions of the main subjects:

- **Git User**: The Git User is the user of the git software.
- **Git UI + Software**: The Git UI could be the terminal of the user's current operating system or a graphical user interface written by a third party, such as GIT Bash. The git software is the actual part, which is installed in the user's computer. It contains all the logical operations required for the git commands.

1.3 Outlook

This is a list of git commands that are still left to model to complete the git manual in such a way so that its completed. However, this list does not contain every git command that is missing as each git command is heavily overloaded. An intrinsically complete manual is not only confusing, but unnecessary too as some overloaded git commands can be replaced with a sequence of other git commands.

- git config - Used to setup a profile for the git user, who contributes something to the
- .gitignore: not a git command. However, it is important for the setup as the file is used to ignore unrelated files from the repository, e.g., .DS_Store
- git diff: To see what has been modified which hasn't been staged yet
- git rm: removes a file from the git repository
- git log: views all commits
- git branch -d and git branch -D: deletes the targeted branch
- git cherry-pick: take a commit from a different branch and reintroduce it as new commit in the current branch
- git revert: take a commit from the current branch, remove it and introduce the removal as new commit. This essentially undoes the changes of the targeted commit.
- git stash: saves modified files on the computers stack. This is not a commit.
- git stash apply: Applies the modifications stored in the stash on the current branch again
- git stash list: Lists out all of the applicable stashes.
- git stash drop: Deletes the stashed modification from the stack.
- git reset: resets the HEAD pointer to a specific state.
- git push –set-upstream: pushes branches that are non existent in the remote repository into the remote repository.

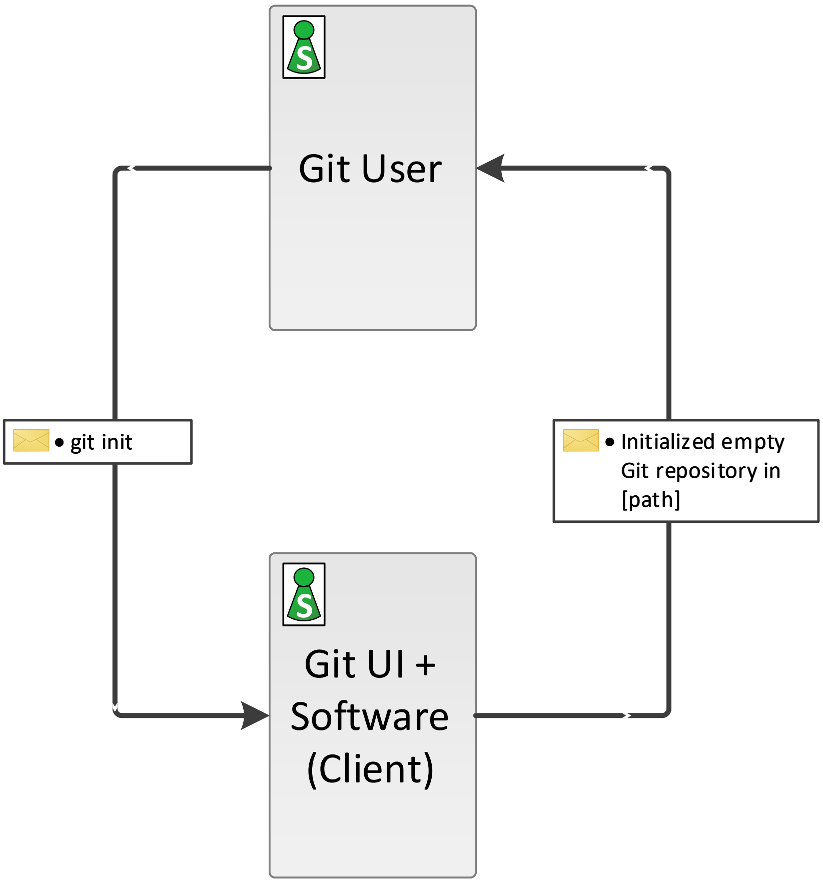
After this list of commands is modeled completely, almost every functionality of git is unlocked and supported by this manual. The commands, which are intentionally left out, are rarely used even by professional software developers. However, if a git expert considers some commands missing, please refer to [CS20] for guidance.

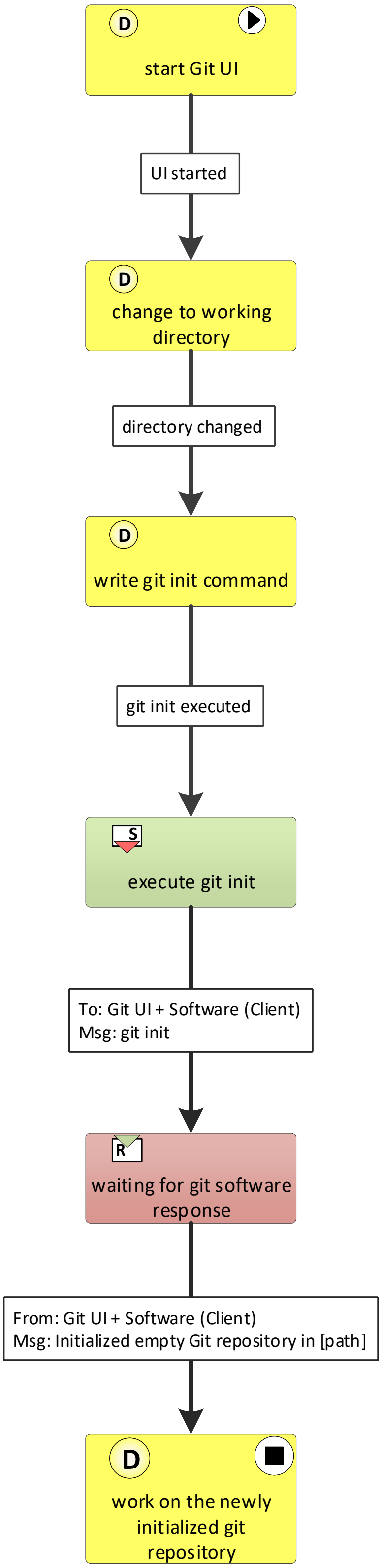
2 Git Basics

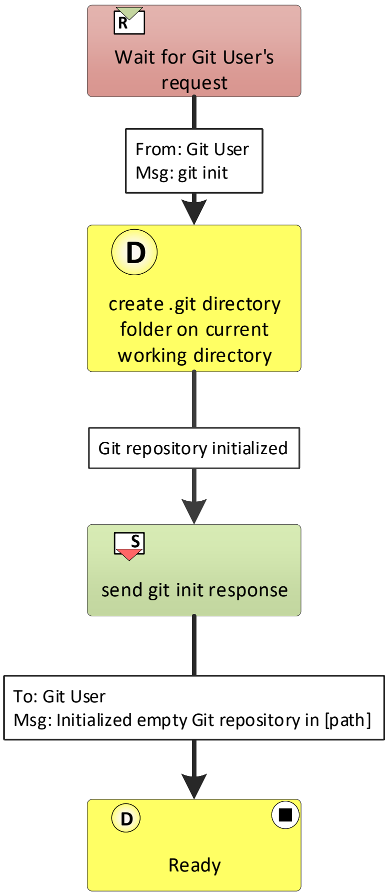
This chapter has every git command related to the basic VCS functionalities. After completing this chapter, the reader will be able to initialize and setup his own git repository, do basic versioning via committing changes and view the current status of the reader’s current working directory.

Additional Information: The git software uses a so-called "HEAD"-pointer to point at the latest commit of the git repository. So if a new commit is done, the HEAD will be updated to point at the new commit. Every commit points to its previous commit. An illustration of this is seen in (3.1).

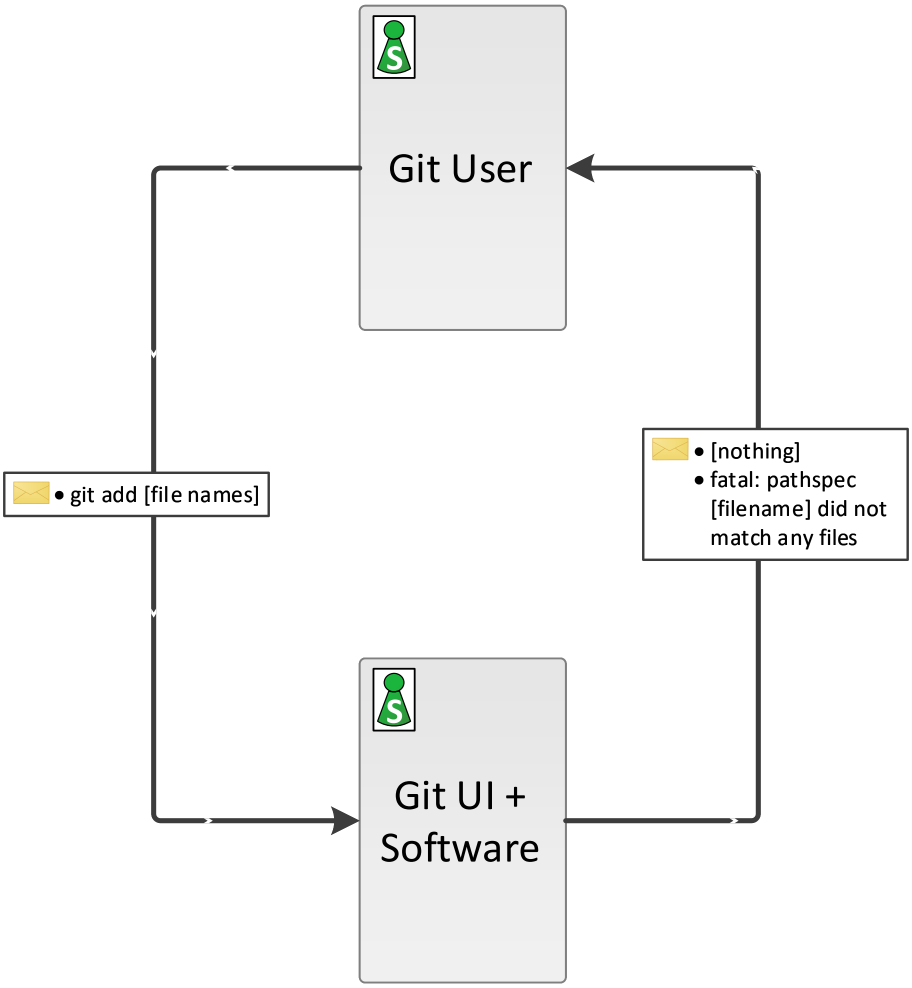
2.1 git init

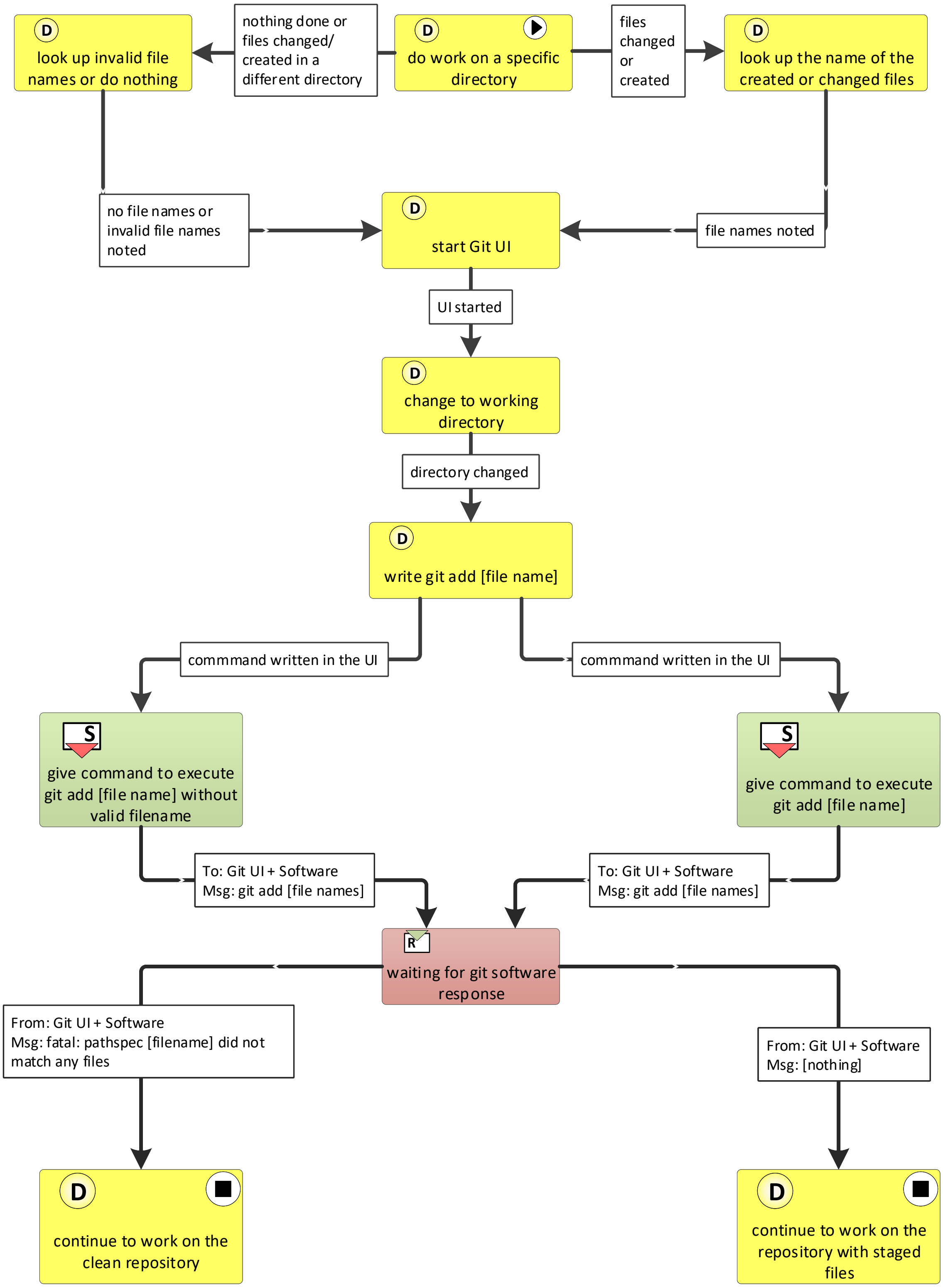

Git User
• git init
• Initialized empty Git repository in [path]
Git UI + Software (Client)

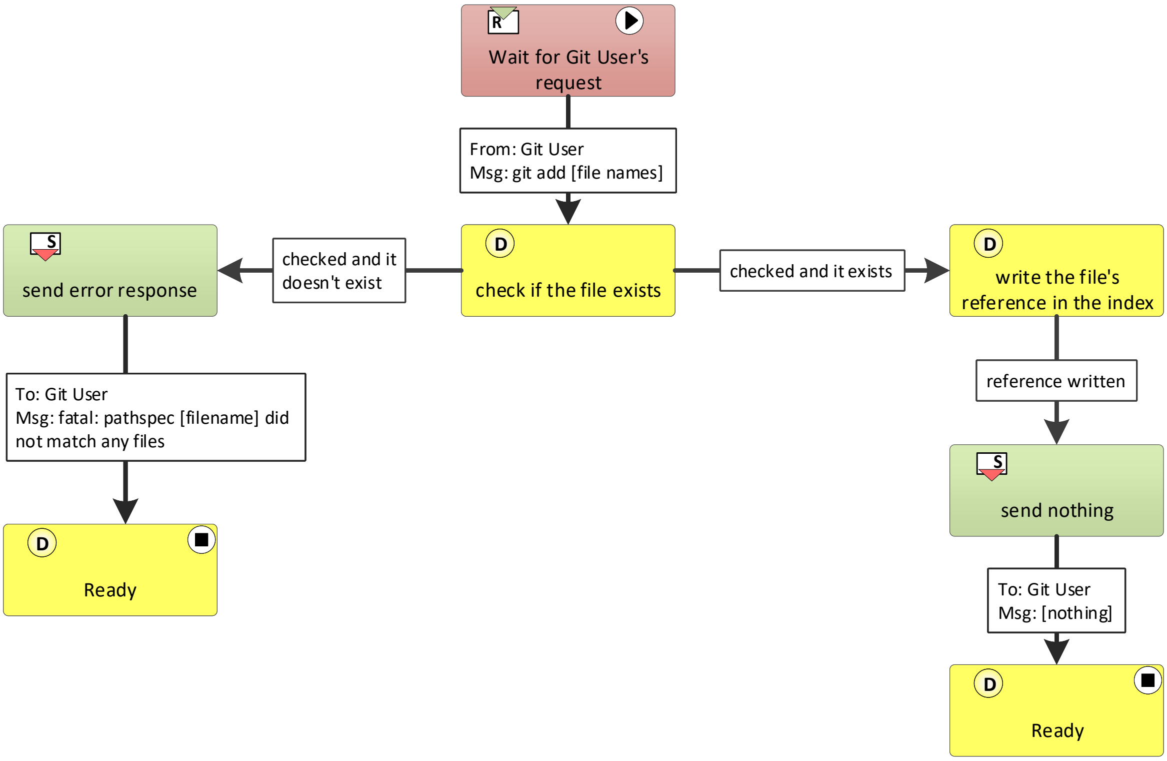

D
start Git UI
UI started
D
change to working directory
directory changed
D
write git init command
git init executed
S
execute git init
To: Git UI + Software (Client)
Msg: git init
R
waiting for git software response
From: Git UI + Software (Client)
Msg: Initialized empty Git repository in [path]
D
work on the newly initialized git repository


R
Wait for Git User's request
From: Git User
Msg: git init
D
create .git directory folder on current working directory
Git repository initialized
S
send git init response
To: Git User
Msg: Initialized empty Git repository in [path]
D
Ready

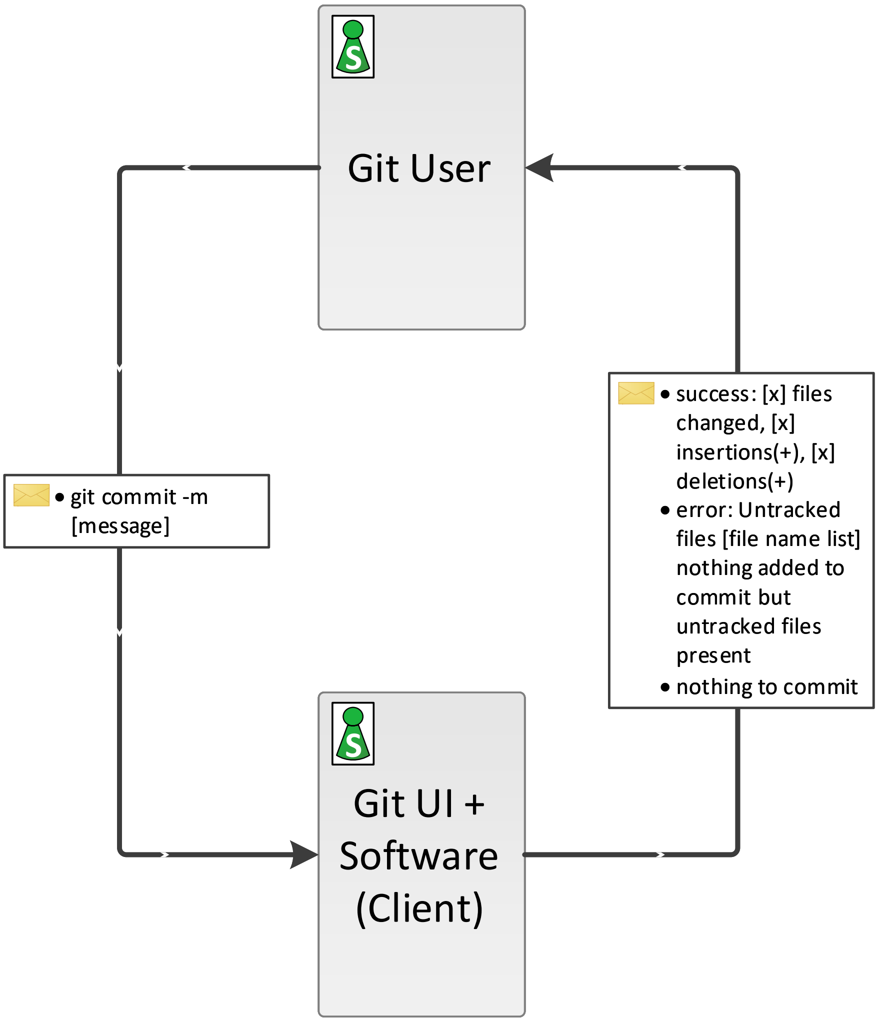
2.2 git add

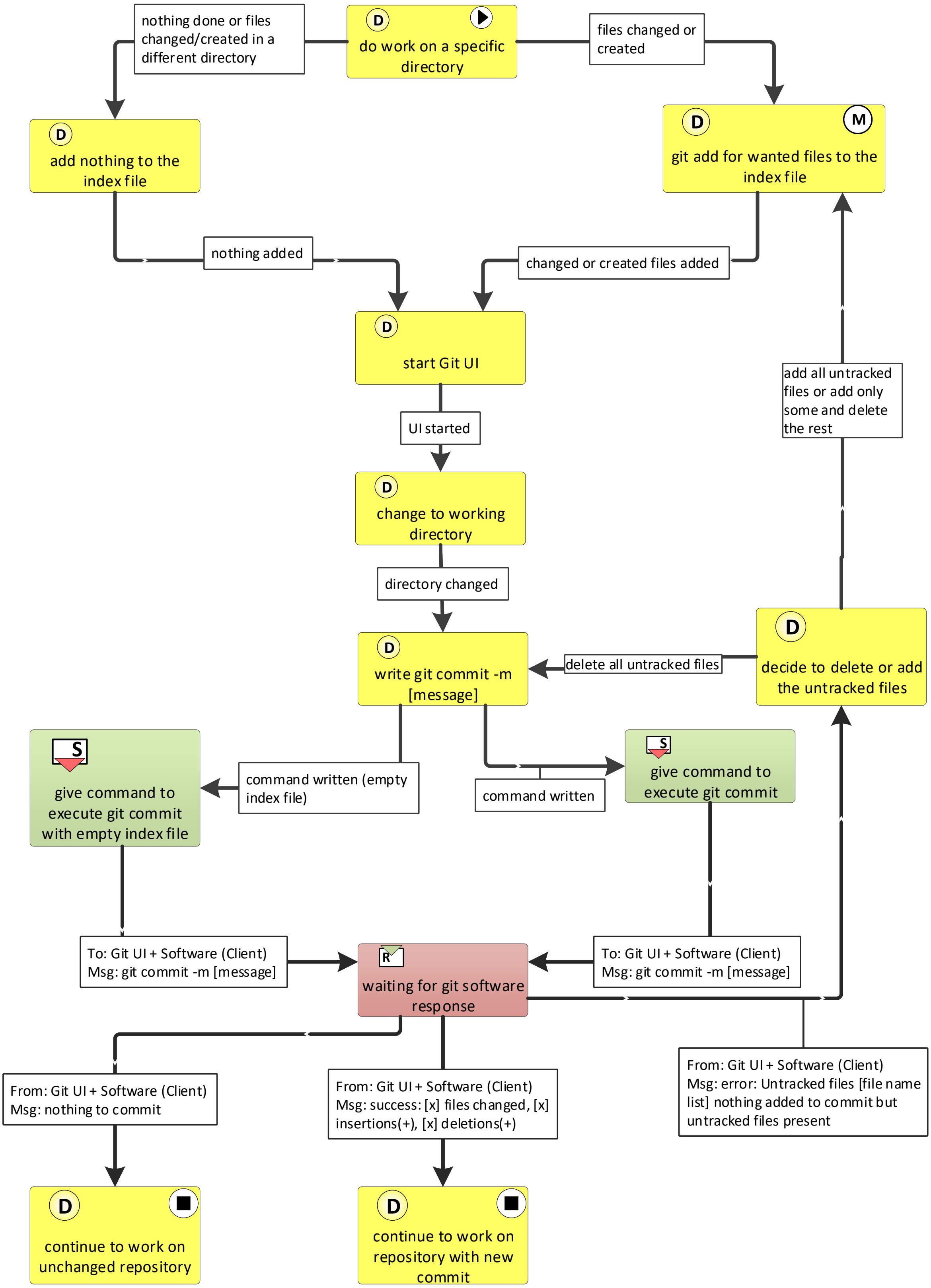

Git User
• git add [file names]
• [nothing]
• fatal: pathspec [filename] did not match any files
Git UI + Software

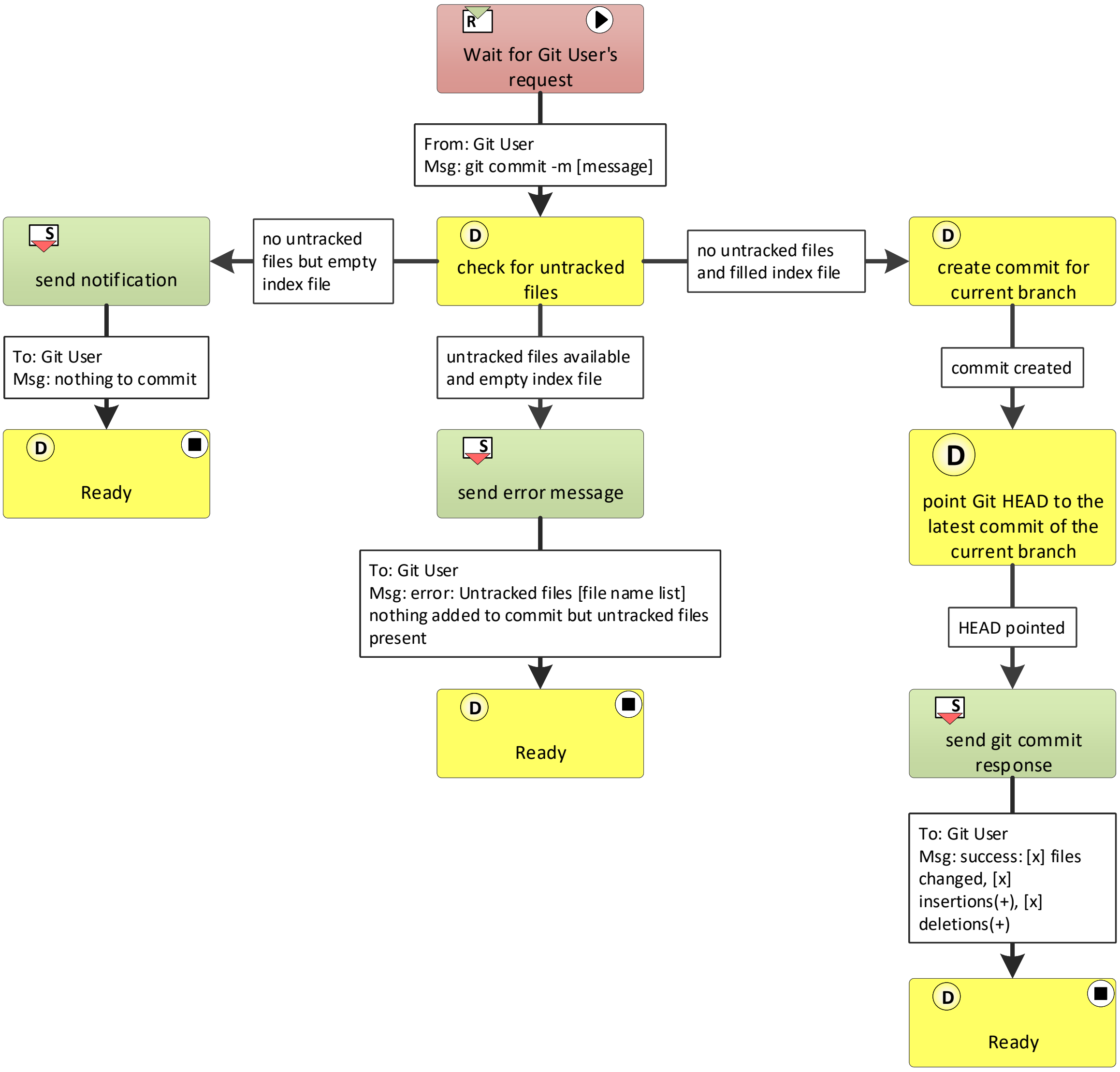

D
look up invalid file names or do nothing
nothing done or files changed/ created in a different directory
D
do work on a specific directory
files changed or created
D
look up the name of the created or changed files
no file names or invalid file names noted
D
start Git UI
file names noted
UI started
D
change to working directory
directory changed
D
write git add [file name]
commmand written in the UI
commmand written in the UI
S
give command to execute git add [file name] without valid filename
S
give command to execute git add [file name]
To: Git UI + Software
Msg: git add [file names]
To: Git UI + Software
Msg: git add [file names]
R
waiting for git software response
From: Git UI + Software
Msg: fatal: pathspec [filename] did not match any files
From: Git UI + Software
Msg: [nothing]
D
continue to work on the clean repository
D
continue to work on the repository with staged files


R
Wait for Git User's request
From: Git User
Msg: git add [file names]
D
check if the file exists
checked and it doesn't exist
S
send error response
To: Git User
Msg: fatal: pathspec [filename] did not match any files
D
Ready
checked and it exists
D
write the file's reference in the index
reference written
S
send nothing
To: Git User
Msg: [nothing]
D
Ready

2.3 git commit

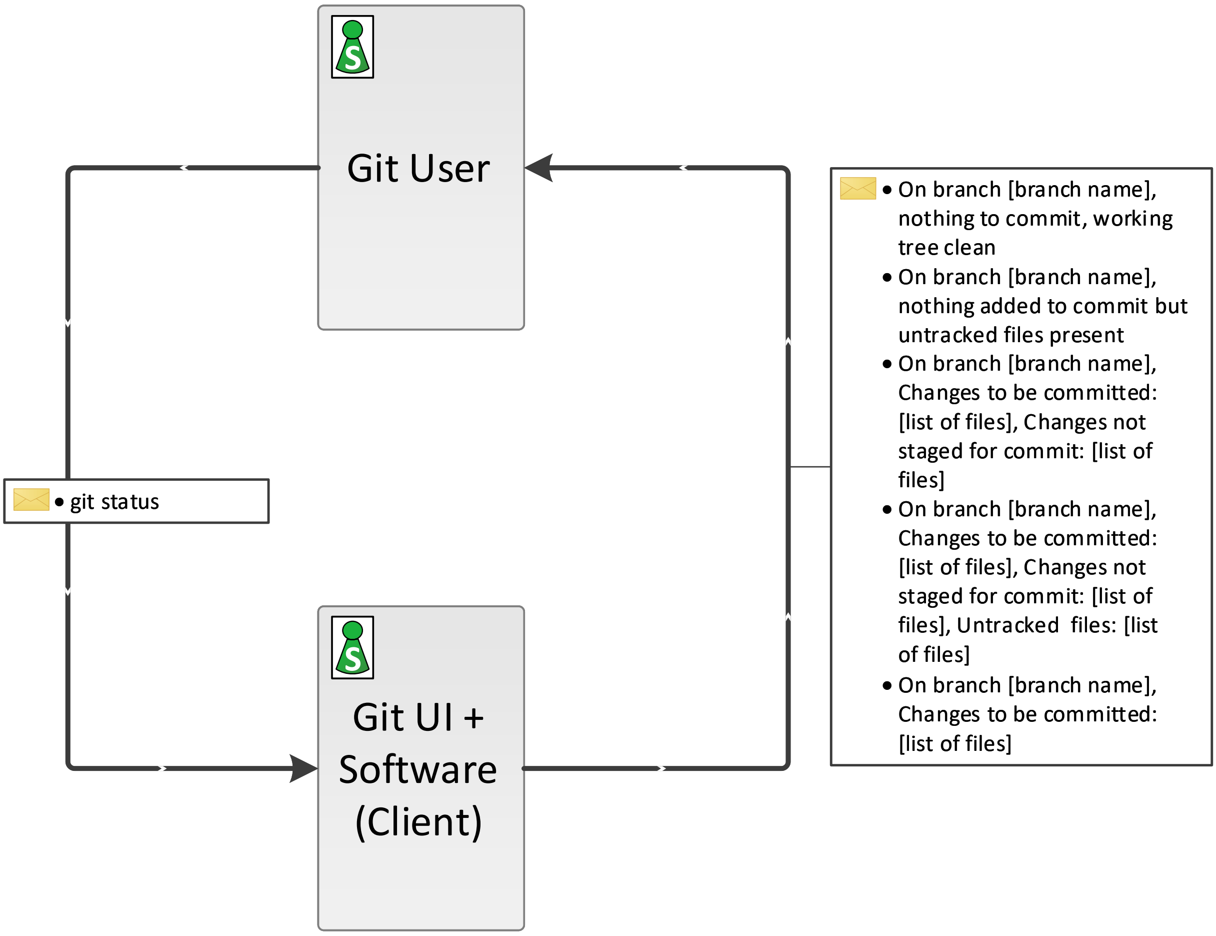

Git User
• git commit -m [message]
• success: [x] files changed, [x] insertions(+), [x] deletions(+)
• error: Untracked files [file name list] nothing added to commit but untracked files present
• nothing to commit
Git UI + Software (Client)

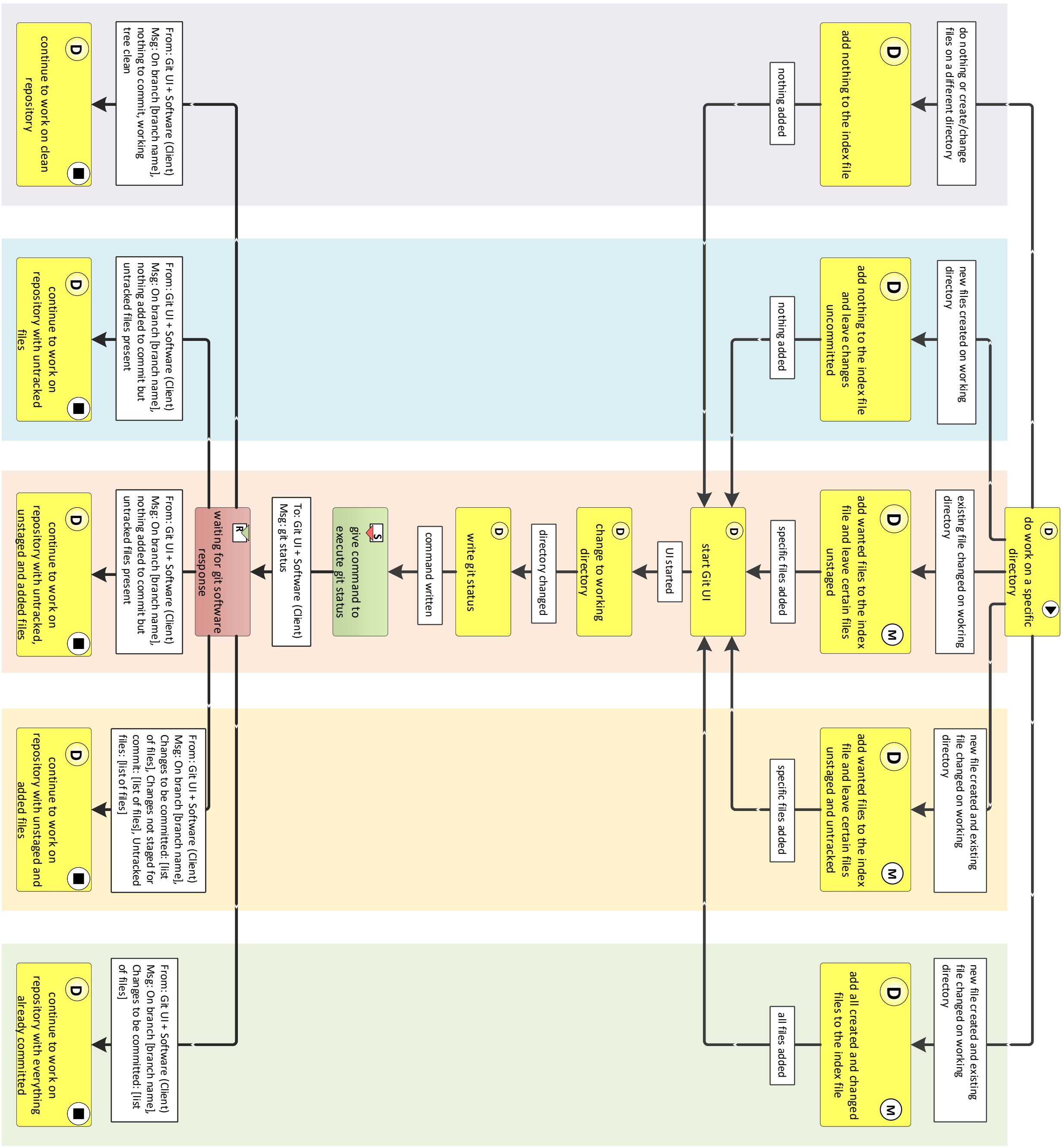

nothing done or files changed/created in a different directory
D
do work on a specific directory
files changed or created
D
add nothing to the index file
D
M
git add for wanted files to the index file
nothing added
changed or created files added
D
start Git UI
add all untracked files or add only some and delete the rest
UI started
D
change to working directory
directory changed
D
write git commit -m [message]
delete all untracked files
D
decide to delete or add the untracked files
S
give command to execute git commit with empty index file
command written (empty index file)
command written
S
give command to execute git commit
To: Git UI + Software (Client)
Msg: git commit -m [message]
R
waiting for git software response
To: Git UI + Software (Client)
Msg: git commit -m [message]
From: Git UI + Software (Client)
Msg: nothing to commit
From: Git UI + Software (Client)
Msg: success: [x] files changed, [x] insertions(+), [x] deletions(+)
From: Git UI + Software (Client)
Msg: error: Untracked files [file name list] nothing added to commit but untracked files present
D
continue to work on unchanged repository
D
continue to work on repository with new commit

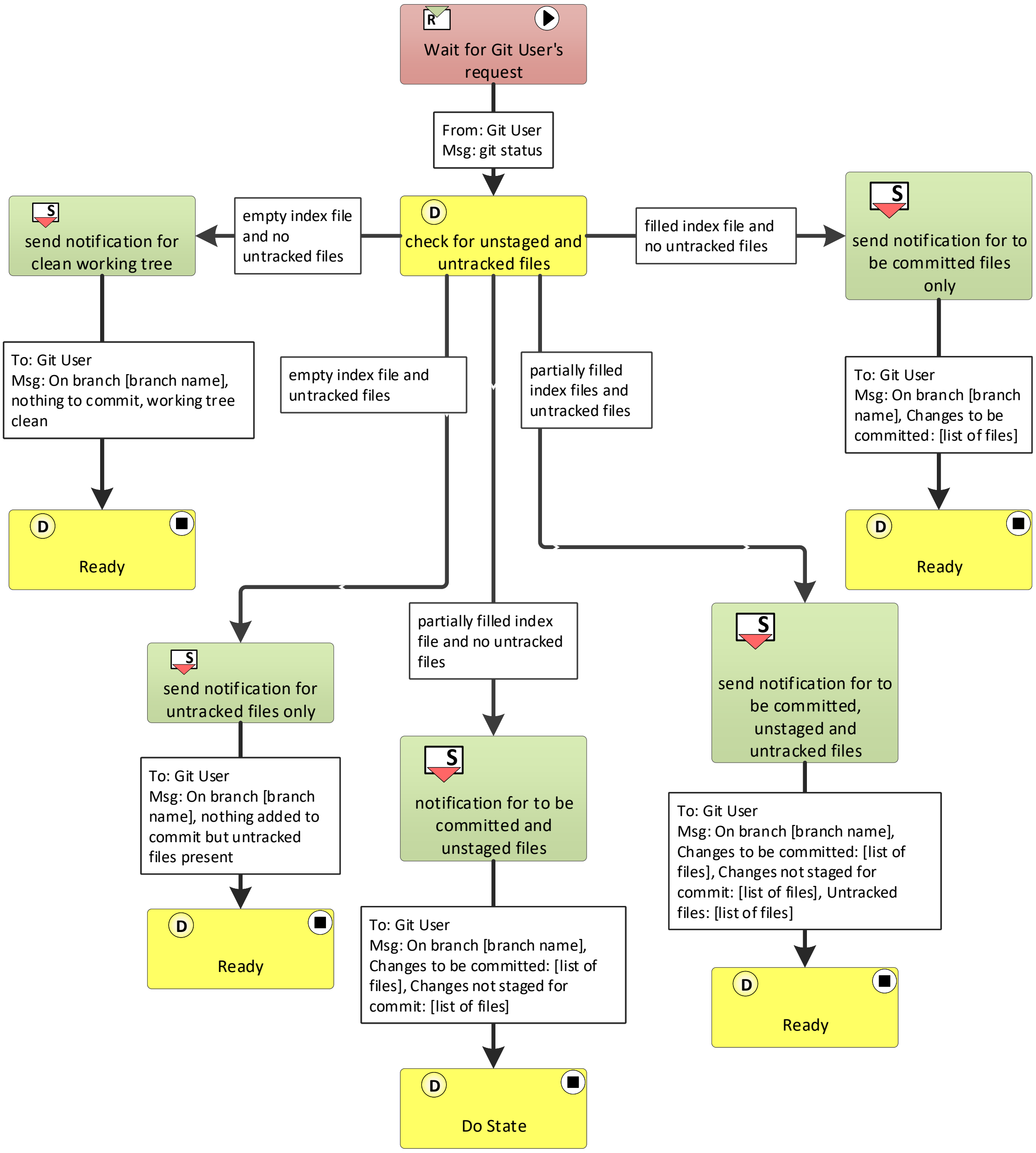

R
Wait for Git User's request
From: Git User
Msg: git commit -m [message]
D
check for untracked files
no untracked files but empty index file
S
send notification
To: Git User
Msg: nothing to commit
D
Ready
untracked files available and empty index file
S
send error message
To: Git User
Msg: error: Untracked files [file name list] nothing added to commit but untracked files present
D
Ready
no untracked files and filled index file
D
create commit for current branch
commit created
D
point Git HEAD to the latest commit of the current branch
HEAD pointed
S
send git commit response
To: Git User
Msg: success: [x] files changed, [x] insertions(+), [x] deletions(+)
D
Ready

2.4 git status

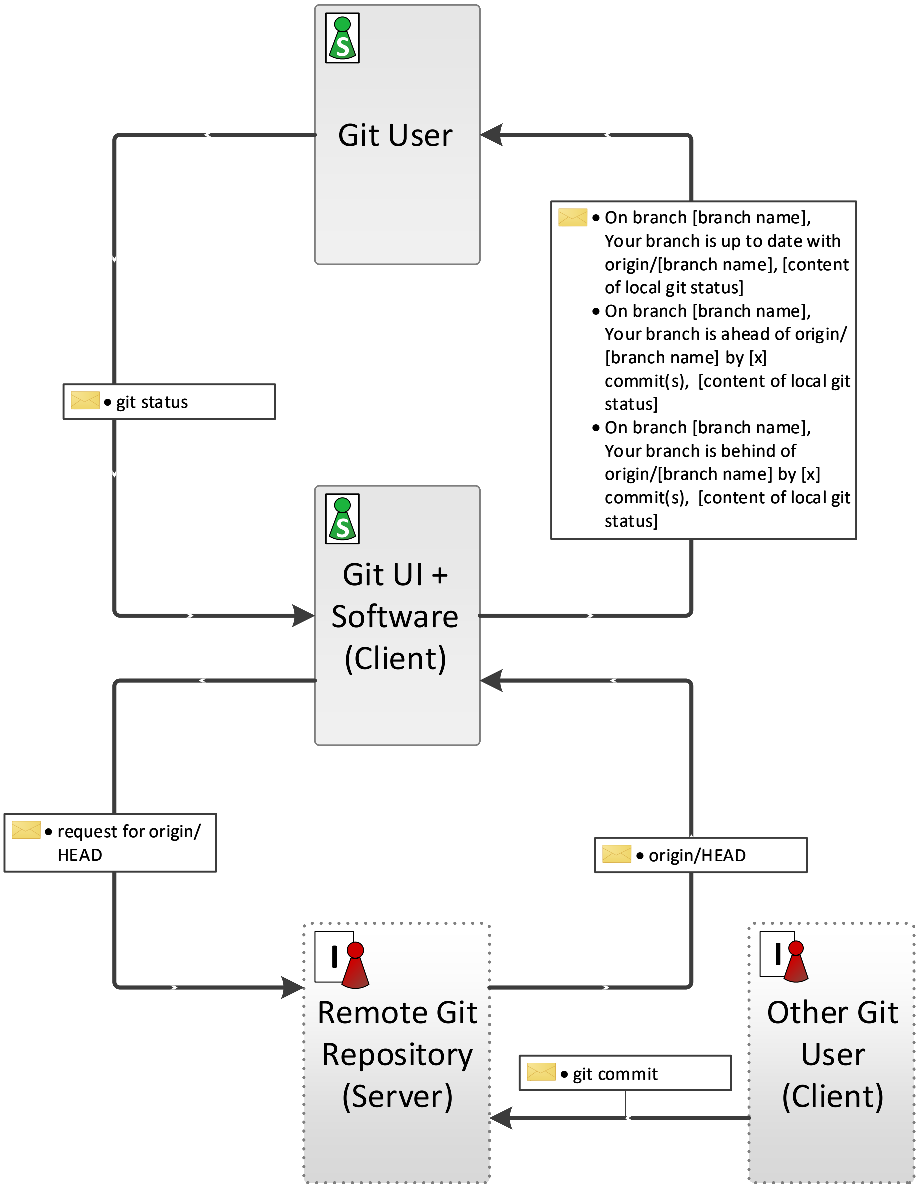
2.4.1 git status local

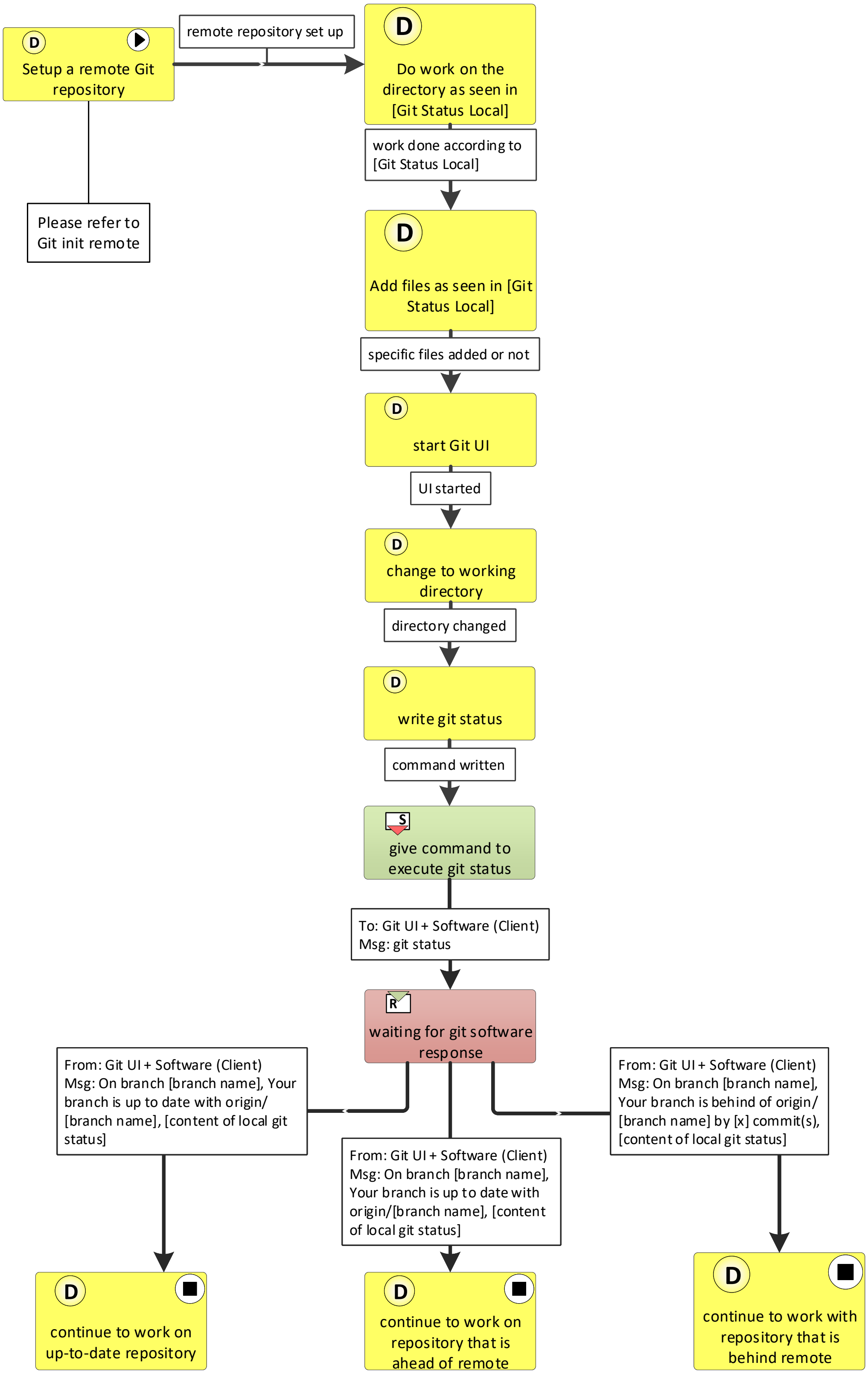

Git User
• git status
Git UI + Software (Client)
• On branch [branch name], nothing to commit, working tree clean
• On branch [branch name], nothing added to commit but untracked files present
• On branch [branch name], Changes to be committed: [list of files], Changes not staged for commit: [list of files]
• On branch [branch name], Changes to be committed: [list of files], Changes not staged for commit: [list of files], Untracked files: [list of files]
• On branch [branch name], Changes to be committed: [list of files]

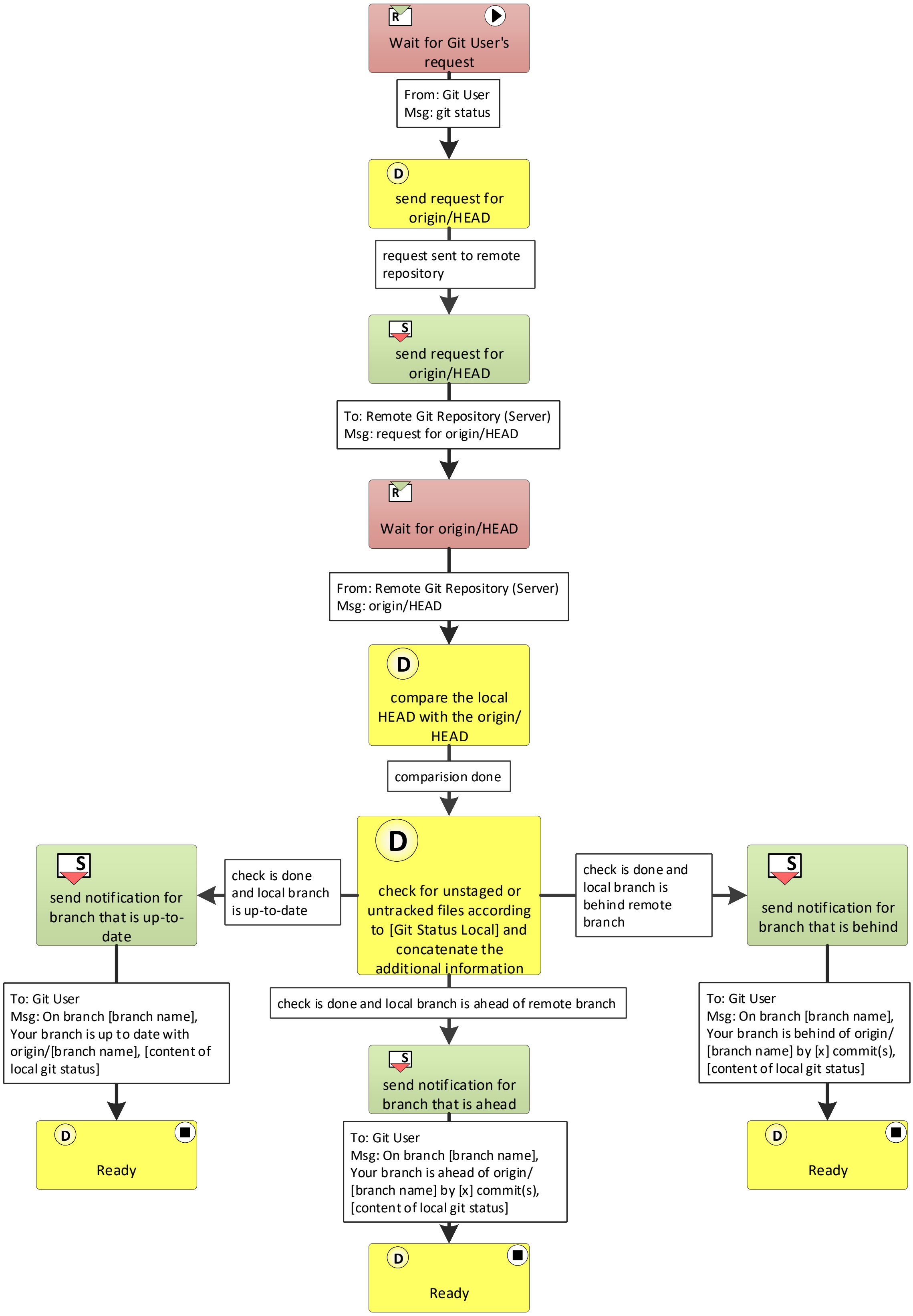

D
do work on a specific directory
do nothing or create/change files on a different directory
new files created on working directory
existing file changed on working directory
new file created and existing file changed on working directory
new file created and existing file changed on working directory
D
add nothing to the index file
D
add nothing to the index file and leave changes uncommitted
D
M
add wanted files to the index file and leave certain files unstaged
D
M
add wanted files to the index file and leave certain files unstaged and untracked
D
M
add all created and changed files to the index file
nothing added
nothing added
specific files added
specific files added
all files added
D
start Git UI
UI started
D
change to working directory
directory changed
D
write git status
command written
S
give command to execute git status
To: Git UI + Software (Client)
Msg: git status
R
waiting for git software response
From: Git UI + Software (Client)
Msg: On branch [branch name], nothing to commit, working tree clean
From: Git UI + Software (Client)
Msg: On branch [branch name], nothing added to commit but untracked files present
From: Git UI + Software (Client)
Msg: On branch [branch name], nothing added to commit but untracked files present
From: Git UI + Software (Client)
Msg: On branch [branch name], Changes to be committed: [list of files], Changes not staged for commit: [list of files], Untracked files: [list of files]
From: Git UI + Software (Client)
Msg: On branch [branch name], Changes to be committed: [list of files]
D
continue to work on clean repository
D
continue to work on repository with untracked files
D
continue to work on repository with untracked, unstaged and added files
D
continue to work on repository with unstaged and added files
D
continue to work on repository with everything already committed


R
Wait for Git User's request
From: Git User
Msg: git status
D
check for unstaged and untracked files
empty index file and no untracked files
S
send notification for clean working tree
To: Git User
Msg: On branch [branch name], nothing to commit, working tree clean
D
Ready
filled index file and no untracked files
S
send notification for to be committed files only
To: Git User
Msg: On branch [branch name], Changes to be committed: [list of files]
D
Ready
empty index file and untracked files
S
send notification for untracked files only
To: Git User
Msg: On branch [branch name], nothing added to commit but untracked files present
D
Ready
partially filled index file and no untracked files
S
notification for to be committed and unstaged files
To: Git User
Msg: On branch [branch name], Changes to be committed: [list of files], Changes not staged for commit: [list of files]
D
Do State
partially filled index files and untracked files
S
send notification for to be committed, unstaged and untracked files
To: Git User
Msg: On branch [branch name], Changes to be committed: [list of files], Changes not staged for commit: [list of files], Untracked files: [list of files]
D
Ready

2.4.2 git status remote


Git User
• On branch [branch name], Your branch is up to date with origin/[branch name], [content of local git status]
• On branch [branch name], Your branch is ahead of origin/[branch name] by [x] commit(s), [content of local git status]
• On branch [branch name], Your branch is behind of origin/[branch name] by [x] commit(s), [content of local git status]
• git status
Git UI + Software (Client)
• request for origin/HEAD
• origin/HEAD
Remote Git Repository (Server)
• git commit
Other Git User (Client)


D
Setup a remote Git repository
remote repository set up
D
Do work on the directory as seen in [Git Status Local]
Please refer to Git init remote
work done according to [Git Status Local]
D
Add files as seen in [Git Status Local]
specific files added or not
D
start Git UI
UI started
D
change to working directory
directory changed
D
write git status
command written
S
give command to execute git status
To: Git UI + Software (Client)
Msg: git status
R
waiting for git software response
From: Git UI + Software (Client)
Msg: On branch [branch name], Your branch is up to date with origin/[branch name], [content of local git status]
From: Git UI + Software (Client)
Msg: On branch [branch name], Your branch is behind of origin/[branch name] by [x] commit(s), [content of local git status]
From: Git UI + Software (Client)
Msg: On branch [branch name], Your branch is up to date with origin/[branch name], [content of local git status]
D
continue to work on up-to-date repository
D
continue to work on repository that is ahead of remote
D
continue to work with repository that is behind remote


R
Wait for Git User's request
From: Git User
Msg: git status
D
send request for origin/HEAD
request sent to remote repository
S
send request for origin/HEAD
To: Remote Git Repository (Server)
Msg: request for origin/HEAD
R
Wait for origin/HEAD
From: Remote Git Repository (Server)
Msg: origin/HEAD
D
compare the local HEAD with the origin/ HEAD
comparision done
D
check for unstaged or untracked files according to [Git Status Local] and concatenate the additional information
check is done and local branch is up-to-date
S
send notification for branch that is up-to-date
To: Git User
Msg: On branch [branch name], Your branch is up to date with origin/[branch name], [content of local git status]
D
Ready
check is done and local branch is behind remote branch
S
send notification for branch that is behind
To: Git User
Msg: On branch [branch name], Your branch is behind of origin/ [branch name] by [x] commit(s), [content of local git status]
D
Ready
check is done and local branch is ahead of remote branch
S
send notification for branch that is ahead
To: Git User
Msg: On branch [branch name], Your branch is ahead of origin/ [branch name] by [x] commit(s), [content of local git status]
D
Ready

3 Git Branching

This chapter focuses on the git commands which are responsible for creating and switching between diverging versioning timelines. The timelines can be worked on independently from each other. These diverging versioning timelines are called "branches". After finishing this chapter, the reader is able to view all existing branches, checkout any existing branch, merge different branches and set a new beginning point for any branch.

Additional Information: Every branch has a branch-pointer. This pointer points at the latest commit of the given branch. The main difference between the branch-pointer and the HEAD (defined in Git Basics) is that the HEAD is actually pointing at the branch-pointer of the current branch. Thus, the HEAD is only indirectly pointing at the latest commit which is dependent on the current branch, as seen in (3.1). Every git repository has a "master" branch. This branch is the main branch from which all other branches may originate.

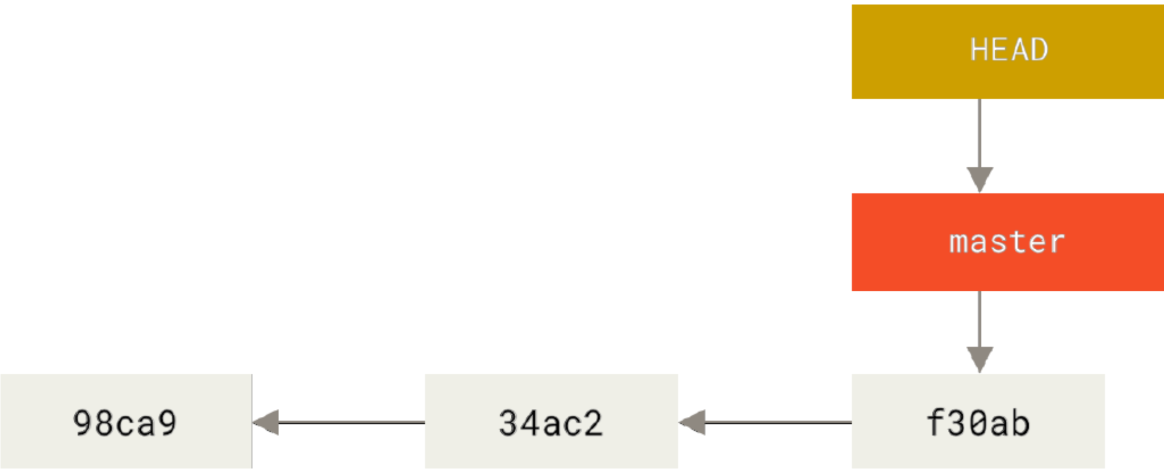
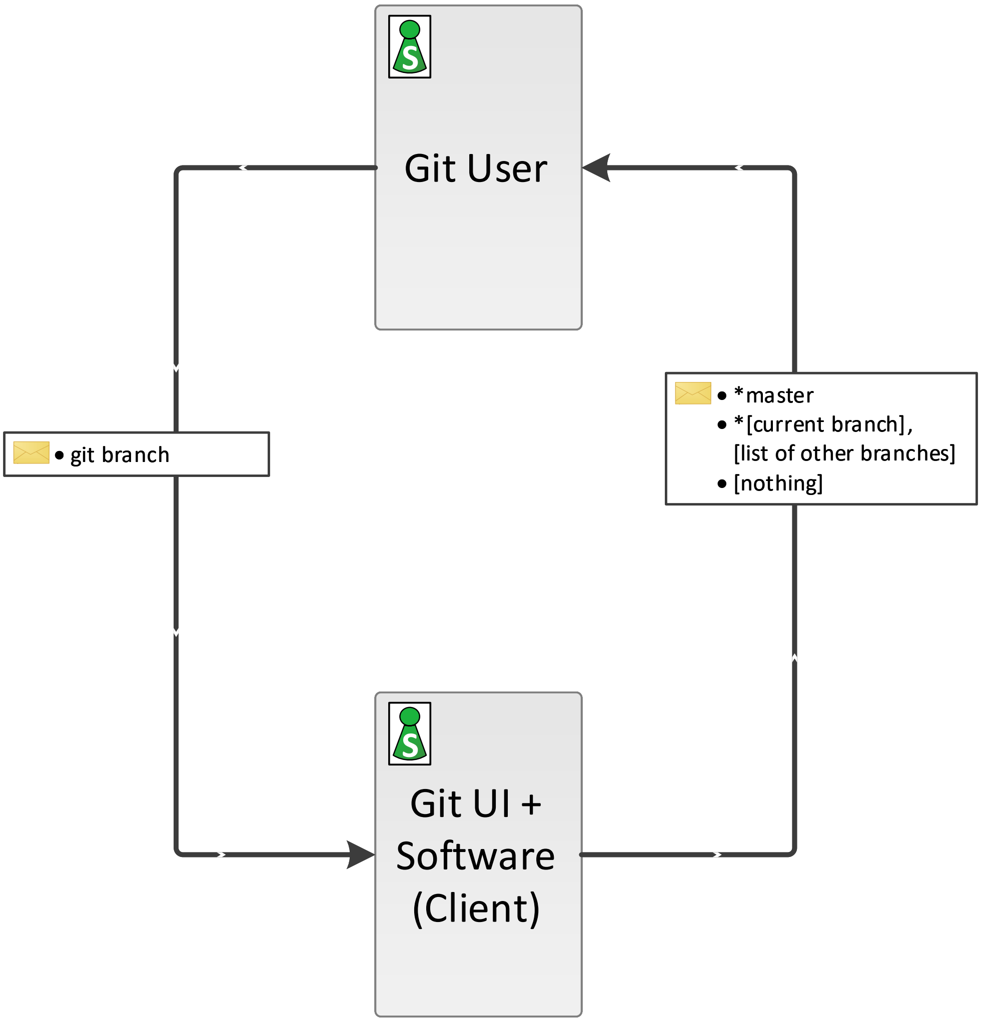
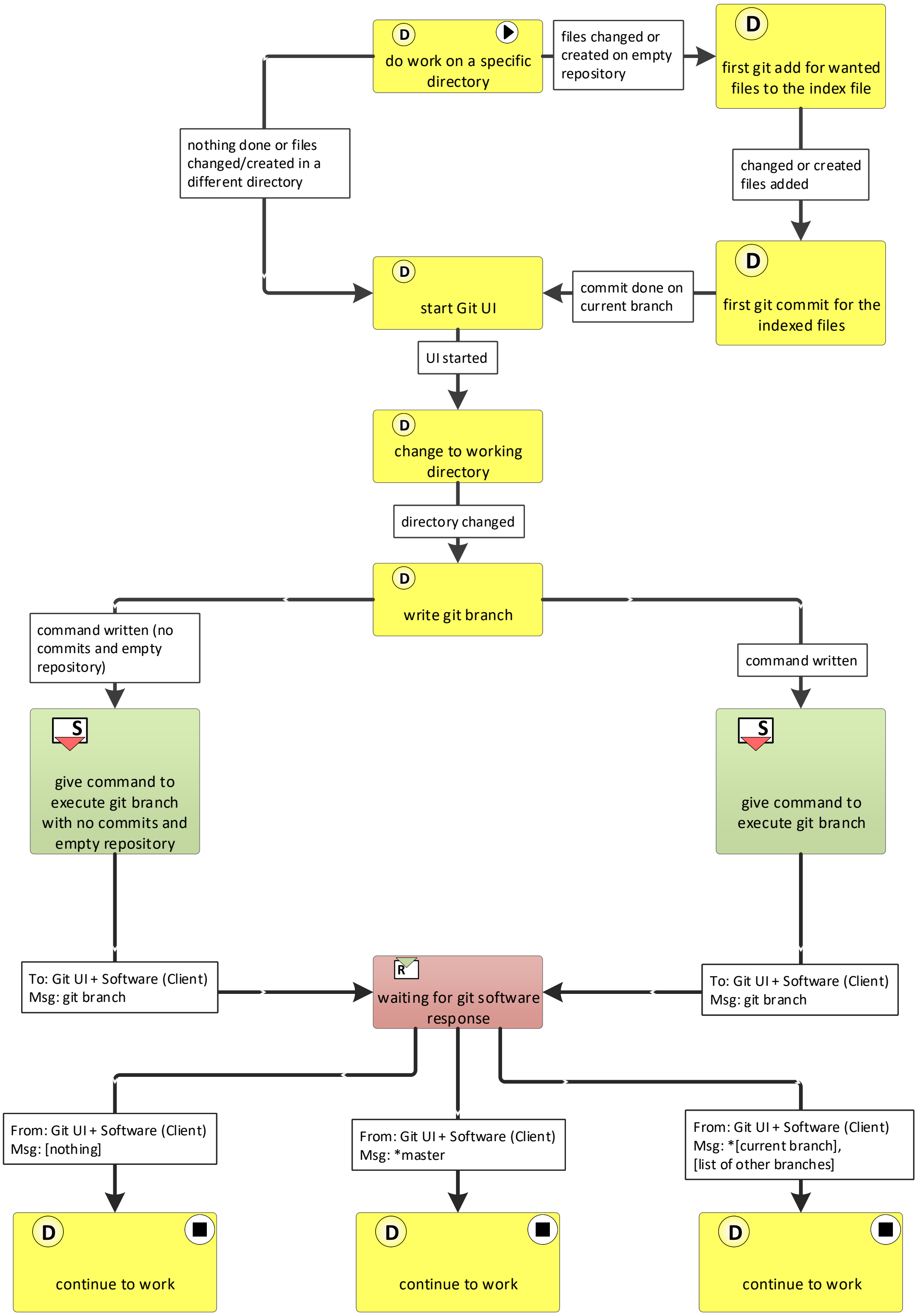


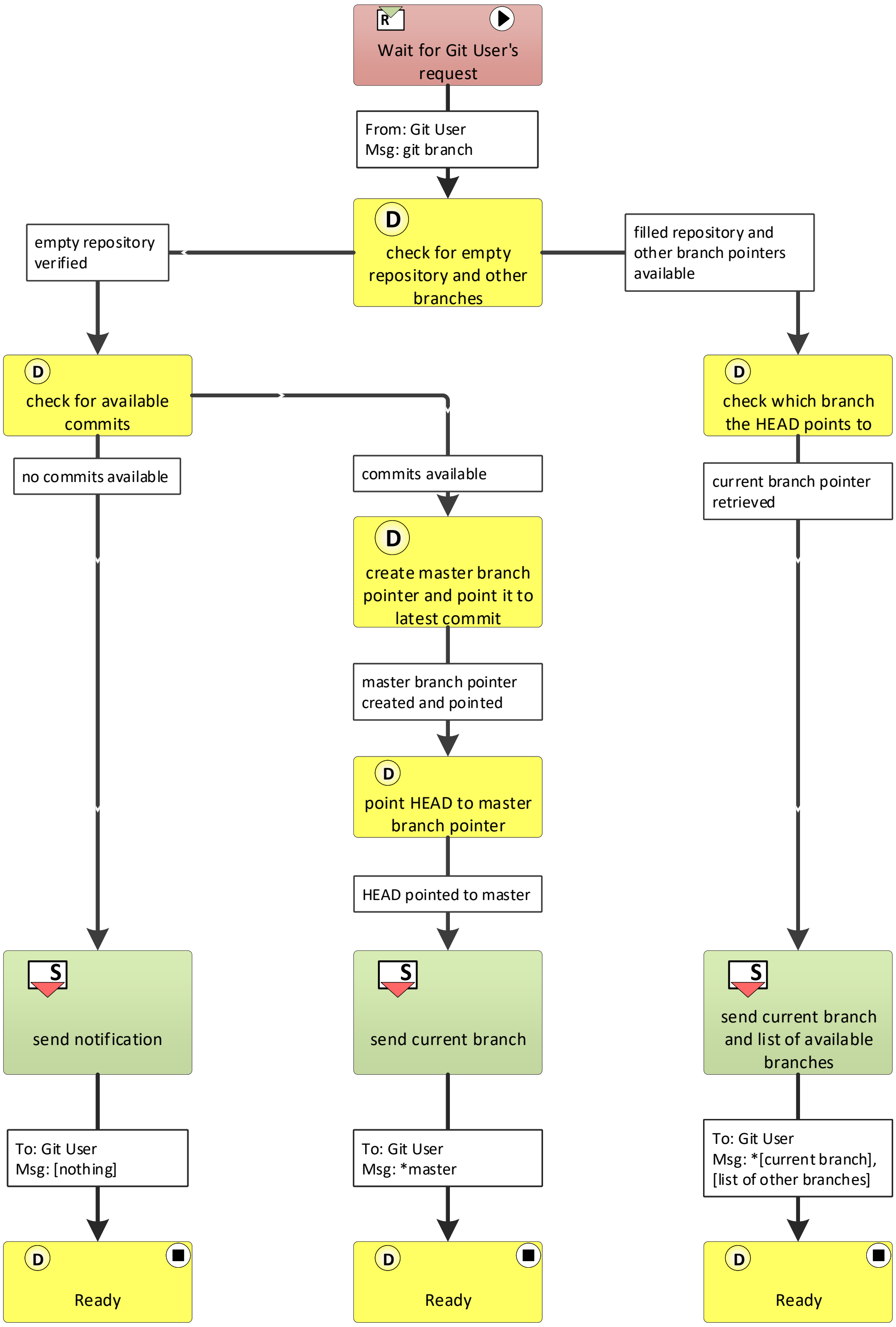
Figure 3.1: The master branch pointer points at the latest commit. The HEAD points at the current branch which is the master [CS20].

3.1 git branch

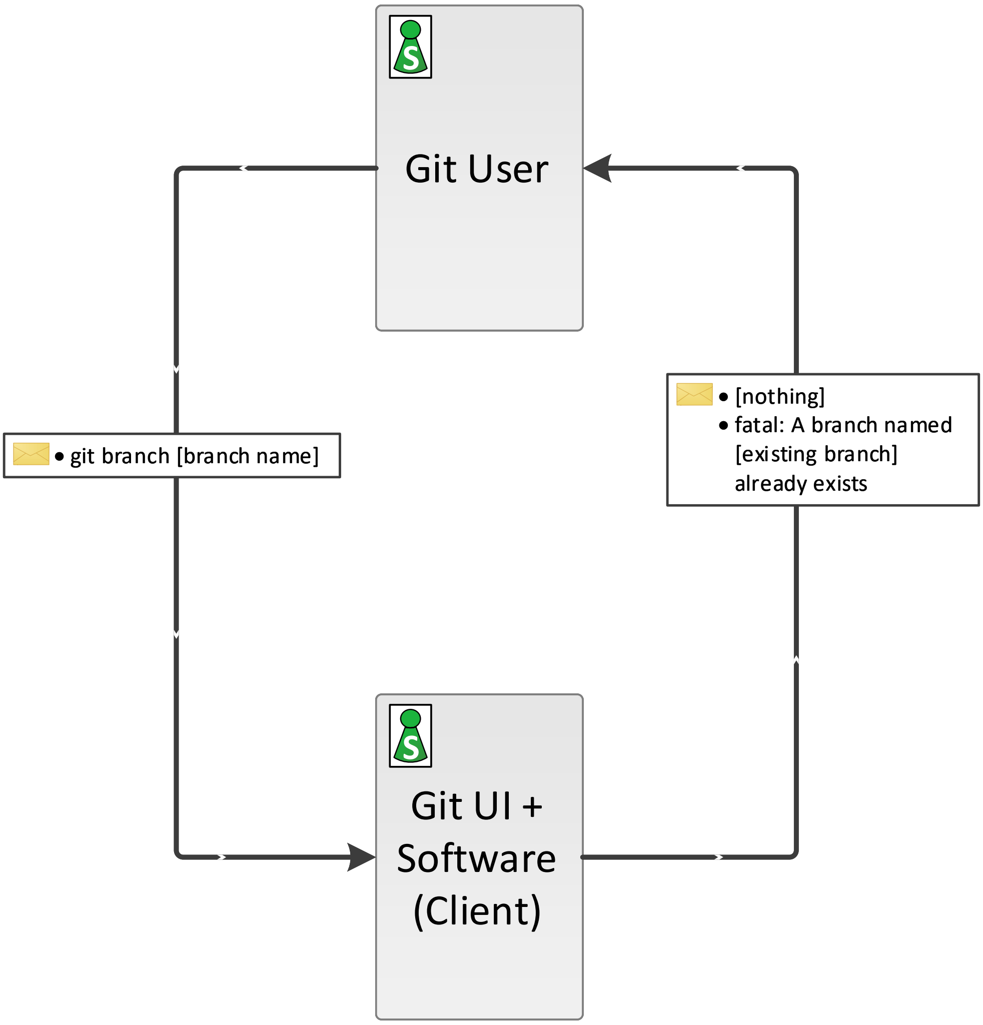
3.1.1 git branch

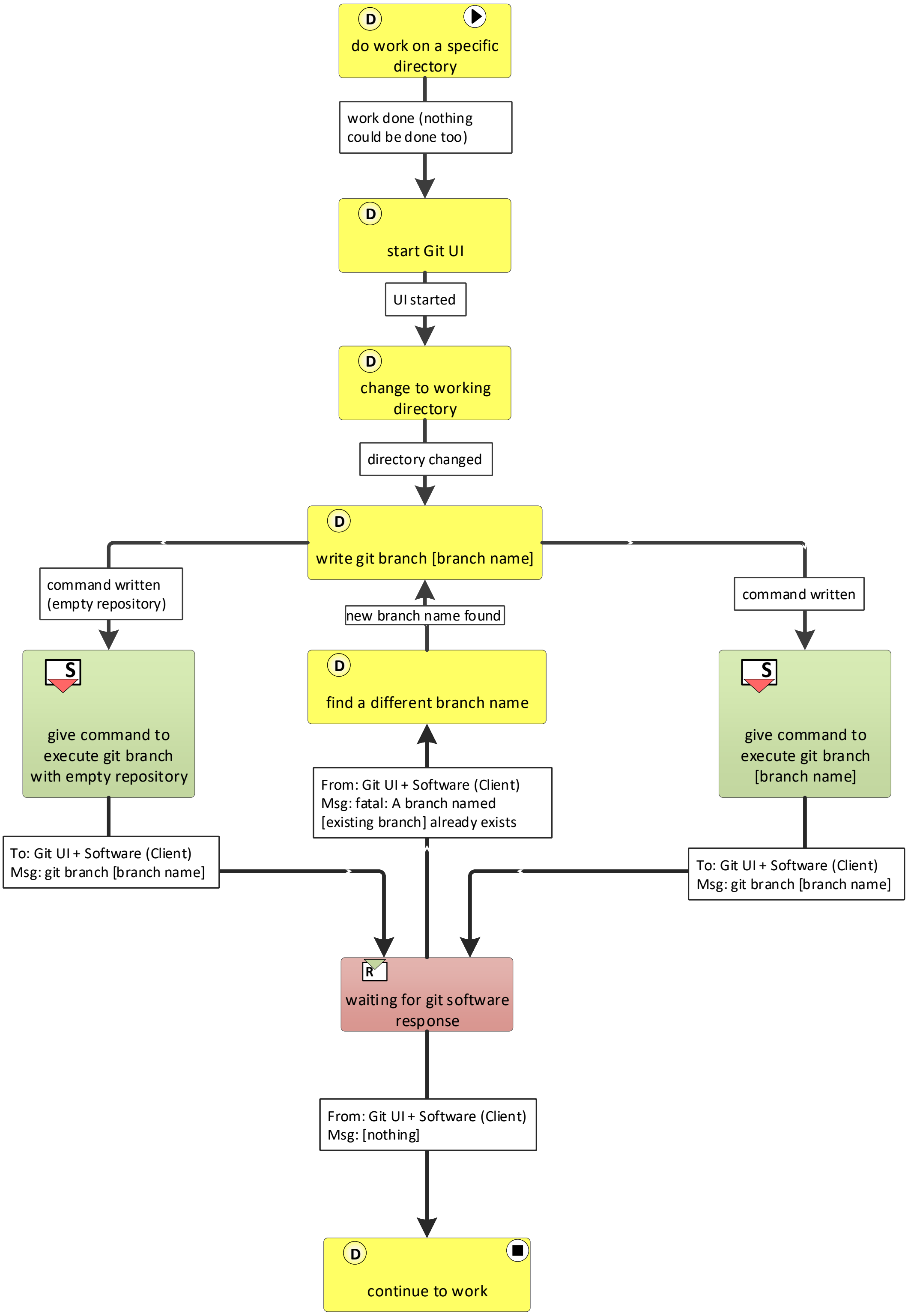


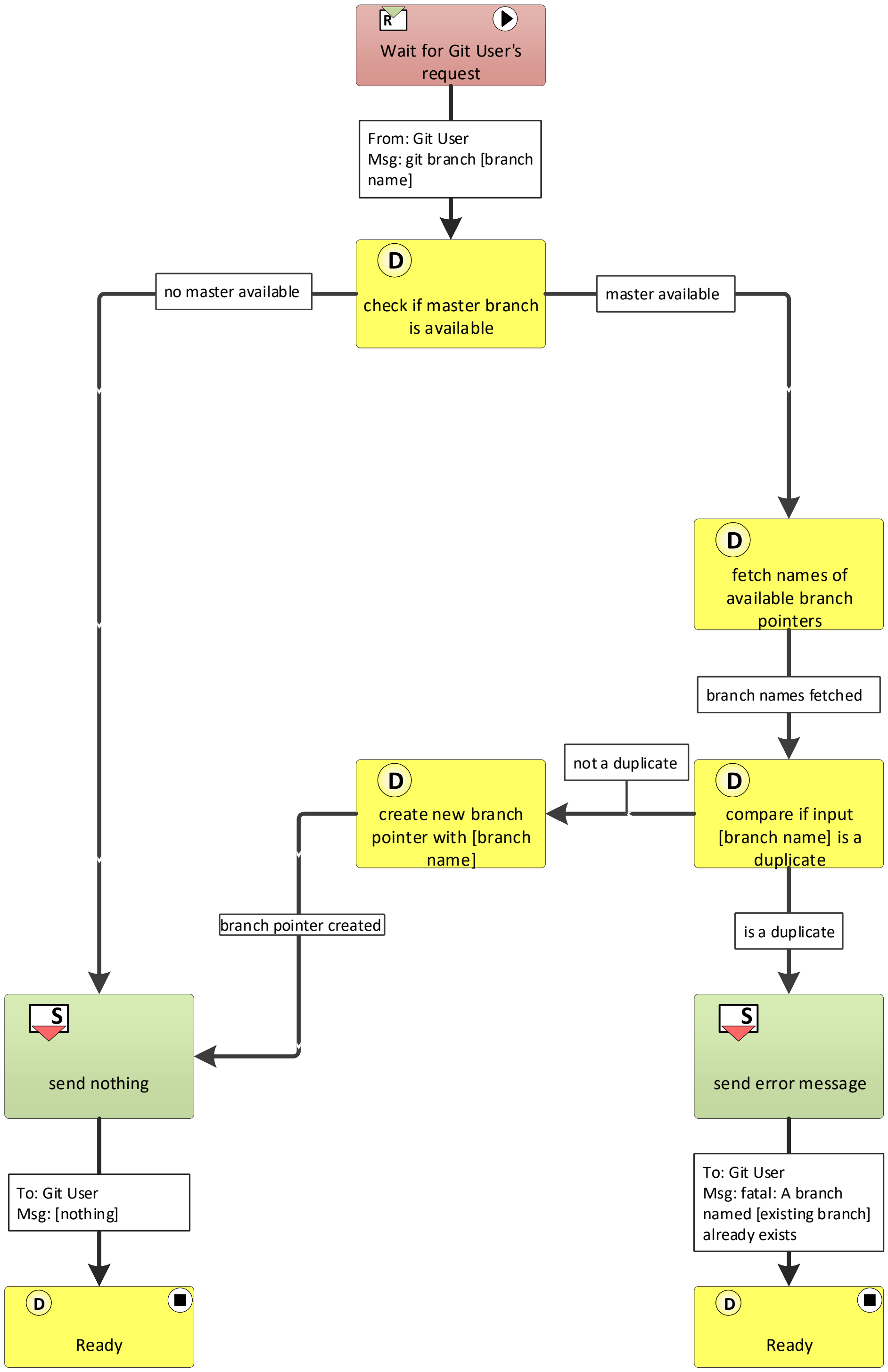

D
do work on a specific directory
files changed or created on empty repository
D
first git add for wanted files to the index file
nothing done or files changed/created in a different directory
changed or created files added
D
start Git UI
commit done on current branch
D
first git commit for the indexed files
UI started
D
change to working directory
directory changed
D
write git branch
command written (no commits and empty repository)
command written
S
give command to execute git branch with no commits and empty repository
S
give command to execute git branch
To: Git UI + Software (Client)
Msg: git branch
R
waiting for git software response
To: Git UI + Software (Client)
Msg: git branch
From: Git UI + Software (Client)
Msg: [nothing]
From: Git UI + Software (Client)
Msg: *master
From: Git UI + Software (Client)
Msg: *[current branch],
[list of other branches]
D
continue to work
D
continue to work
D
continue to work


R
Wait for Git User's request
From: Git User
Msg: git branch
D
check for empty repository and other branches
empty repository verified
filled repository and other branch pointers available
D
check for available commits
no commits available
commits available
D
create master branch pointer and point it to latest commit
master branch pointer created and pointed
D
point HEAD to master branch pointer
HEAD pointed to master
D
check which branch the HEAD points to
current branch pointer retrieved
S
send notification
To: Git User
Msg: [nothing]
S
send current branch
To: Git User
Msg: *master
S
send current branch and list of available branches
To: Git User
Msg: *[current branch], [list of other branches]
D
Ready
D
Ready
D
Ready

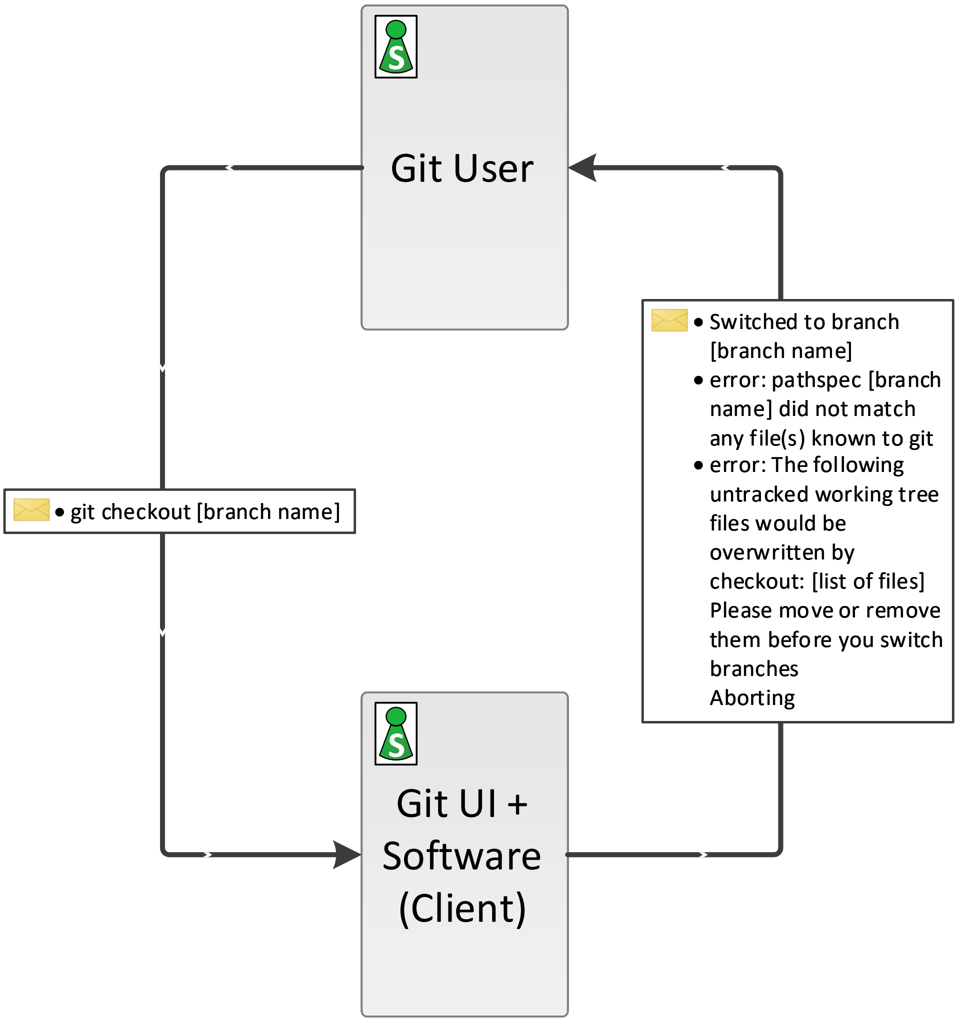
3.1.2 git branch [name]

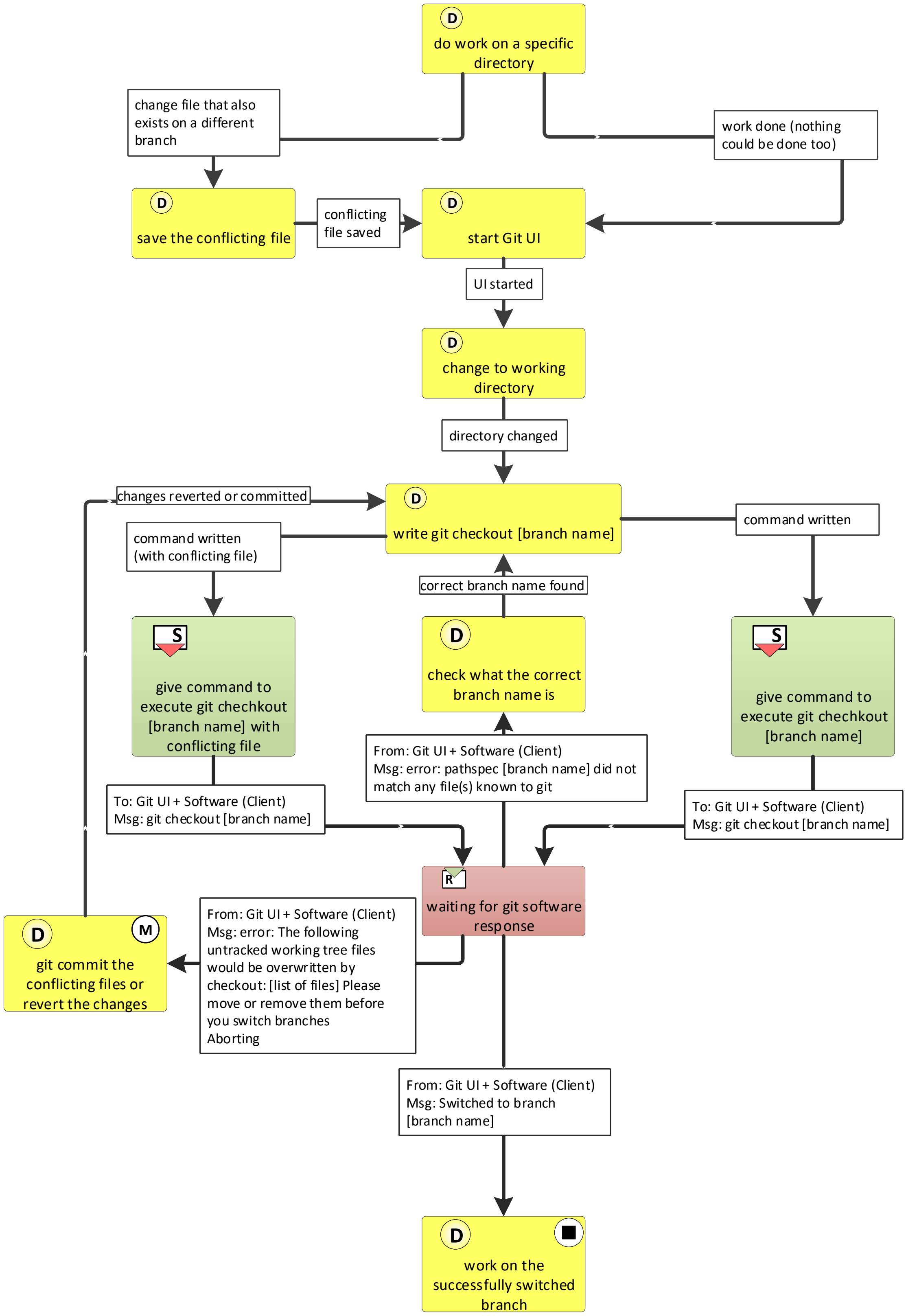

Git User
Git UI + Software (Client)
• git branch [branch name]
• [nothing]
• fatal: A branch named [existing branch] already exists

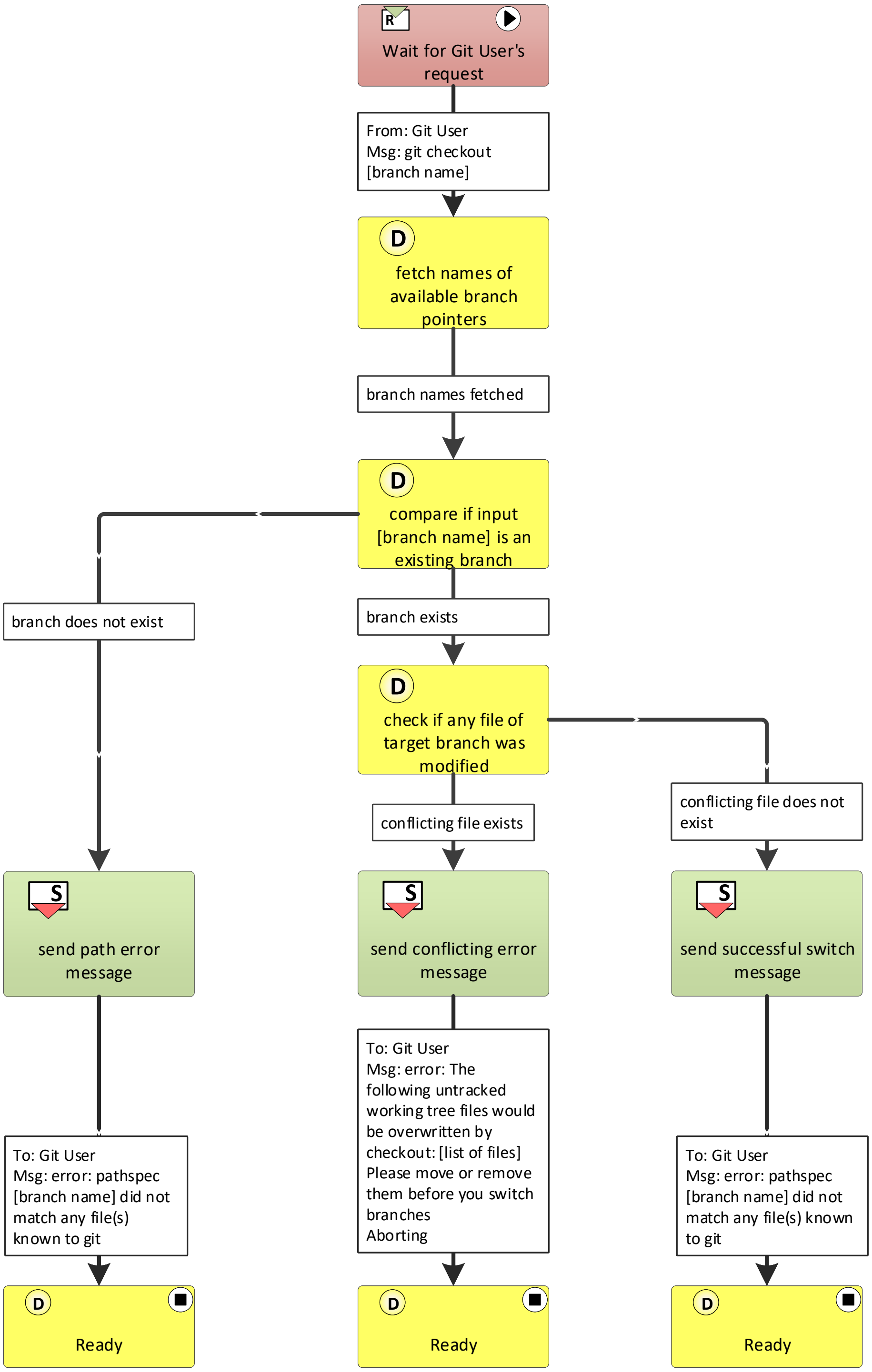

D
do work on a specific directory
work done (nothing could be done too)
D
start Git UI
UI started
D
change to working directory
directory changed
D
write git branch [branch name]
command written (empty repository)
new branch name found
command written
S
give command to execute git branch with empty repository
D
find a different branch name
S
give command to execute git branch [branch name]
From: Git UI + Software (Client)
Msg: fatal: A branch named [existing branch] already exists
To: Git UI + Software (Client)
Msg: git branch [branch name]
To: Git UI + Software (Client)
Msg: git branch [branch name]
R
waiting for git software response
From: Git UI + Software (Client)
Msg: [nothing]
D
continue to work


R
Wait for Git User's request
From: Git User
Msg: git branch [branch name]
D
check if master branch is available
no master available
master available
D
fetch names of available branch pointers
branch names fetched
not a duplicate
D
compare if input [branch name] is a duplicate
D
create new branch pointer with [branch name]
branch pointer created
is a duplicate
S
send nothing
S
send error message
To: Git User
Msg: [nothing]
To: Git User
Msg: fatal: A branch named [existing branch] already exists
D
Ready
D
Ready

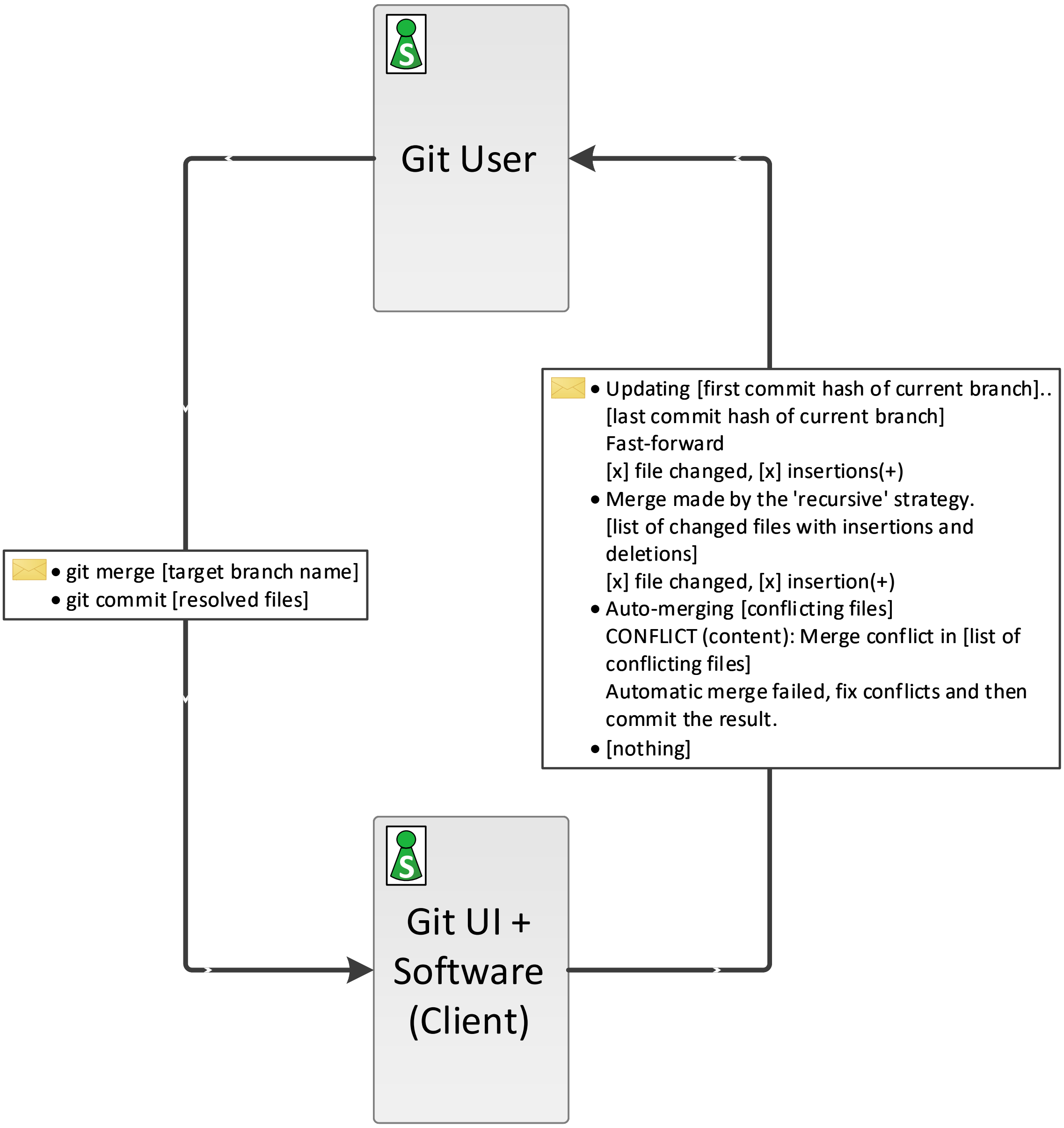
3.2 git checkout


Git User
• git checkout [branch name]
• Switched to branch [branch name]
• error: pathspec [branch name] did not match any file(s) known to git
• error: The following untracked working tree files would be overwritten by checkout: [list of files] Please move or remove them before you switch branches Aborting
Git UI + Software (Client)


D
do work on a specific directory
change file that also exists on a different branch
work done (nothing could be done too)
D
save the conflicting file
conflicting file saved
D
start Git UI
UI started
D
change to working directory
directory changed
changes reverted or committed
D
write git checkout [branch name]
command written (with conflicting file)
command written
correct branch name found
S
give command to execute git chechkout [branch name] with conflicting file
D
check what the correct branch name is
S
give command to execute git chechkout [branch name]
From: Git UI + Software (Client)
Msg: error: pathspec [branch name] did not match any file(s) known to git
To: Git UI + Software (Client)
Msg: git checkout [branch name]
To: Git UI + Software (Client)
Msg: git checkout [branch name]
R
waiting for git software response
D
M
git commit the conflicting files or revert the changes
From: Git UI + Software (Client)
Msg: error: The following untracked working tree files would be overwritten by checkout: [list of files] Please move or remove them before you switch branches
Aborting
From: Git UI + Software (Client)
Msg: Switched to branch [branch name]
D
work on the successfully switched branch

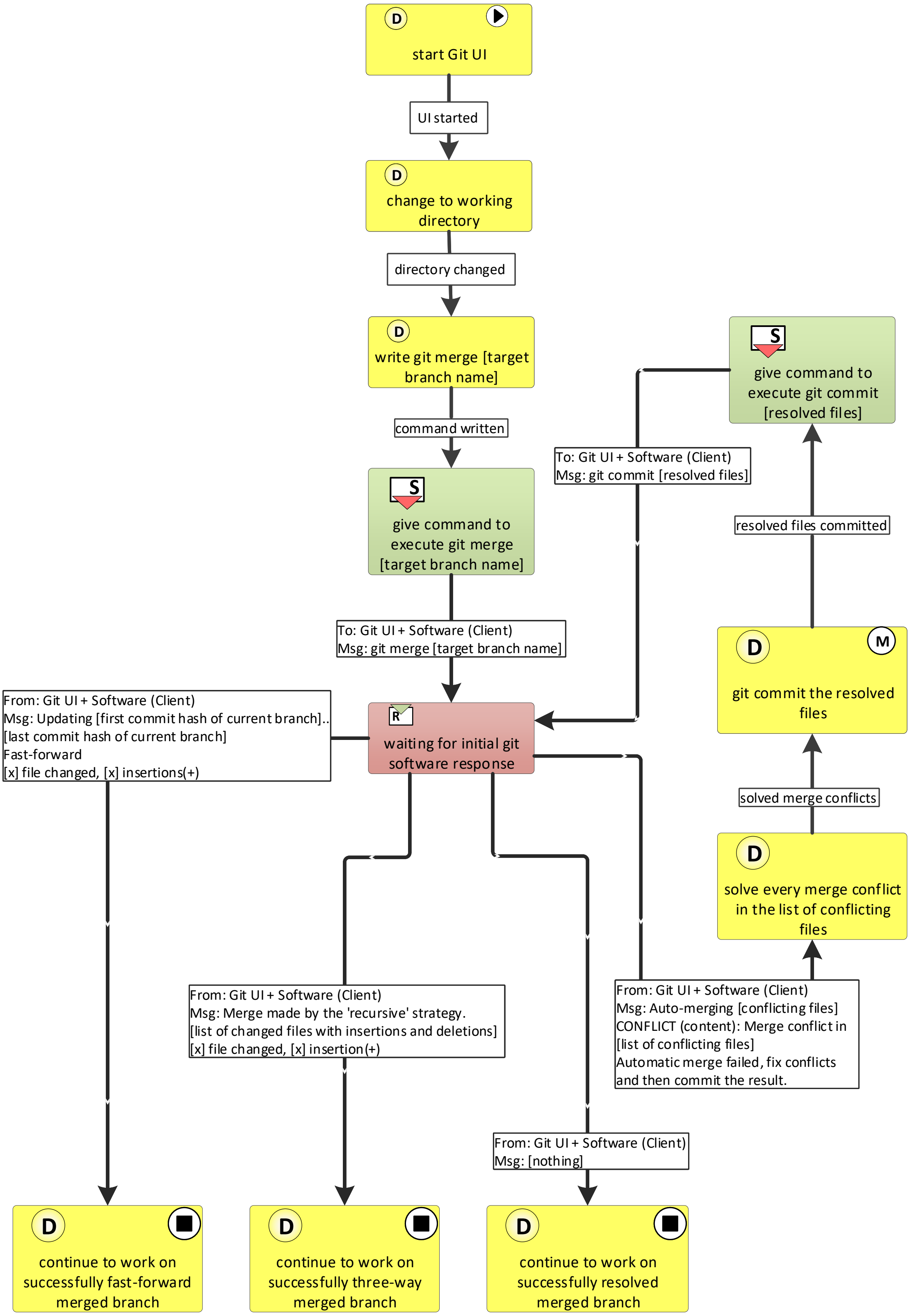

R
Wait for Git User's request
From: Git User
Msg: git checkout [branch name]
D
fetch names of available branch pointers
branch names fetched
D
compare if input [branch name] is an existing branch
branch does not exist
branch exists
D
check if any file of target branch was modified
conflicting file exists
conflicting file does not exist
S
send path error message
S
send conflicting error message
S
send successful switch message
To: Git User
Msg: error: pathspec [branch name] did not match any file(s) known to git
To: Git User
Msg: error: The following untracked working tree files would be overwritten by checkout: [list of files] Please move or remove them before you switch branches
Aborting
To: Git User
Msg: error: pathspec [branch name] did not match any file(s) known to git
D
Ready
D
Ready
D
Ready

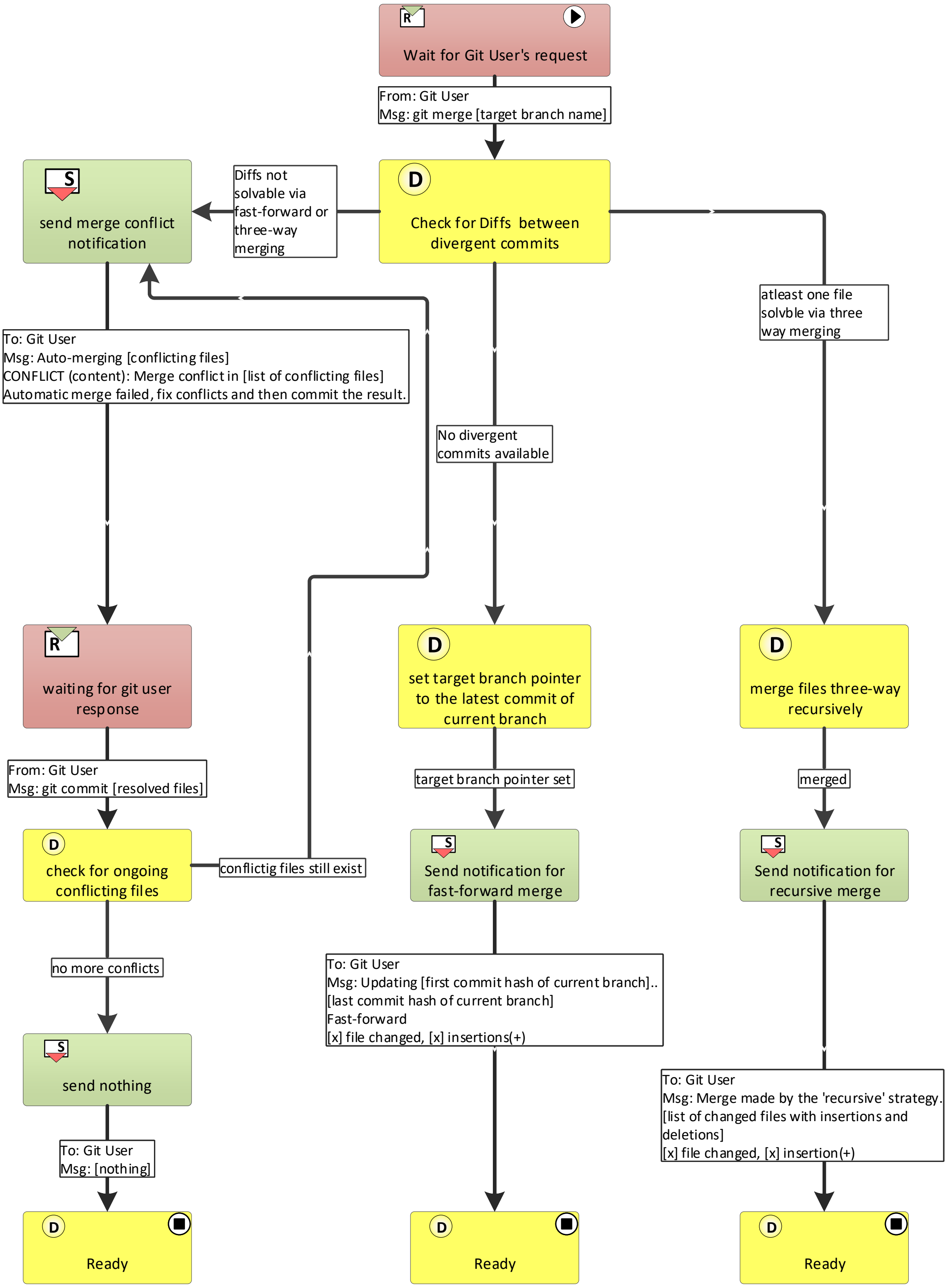
3.3 git merge



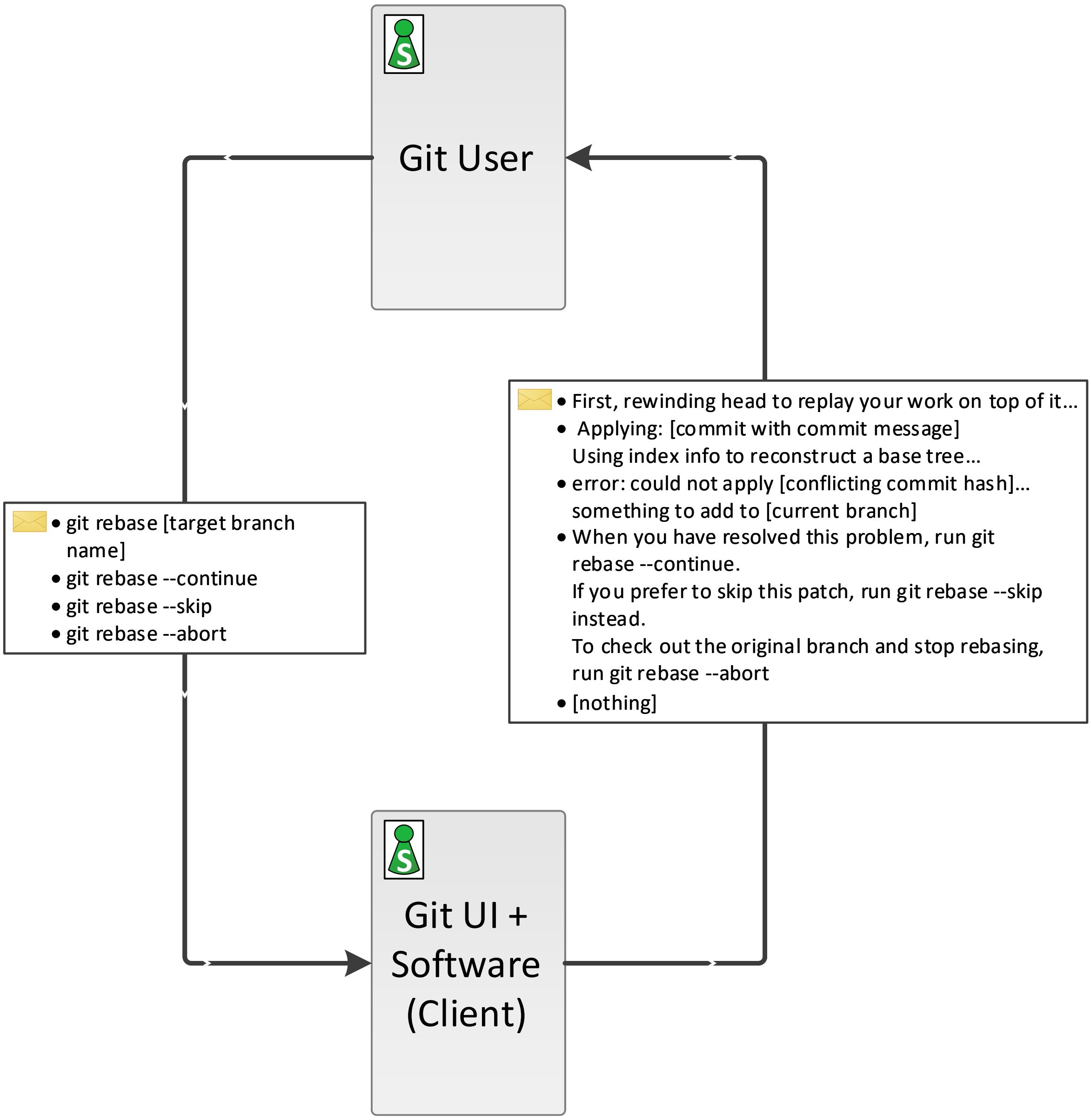
- git merge [target branch name]
- git commit [resolved files]

- Updating [first commit hash of current branch]..
  [last commit hash of current branch]
  Fast-forward
  [x] file changed, [x] insertions(+)
- Merge made by the 'recursive' strategy.
  [list of changed files with insertions and deletions]
  [x] file changed, [x] insertion(+)
- Auto-merging [conflicting files]
  CONFLICT (content): Merge conflict in [list of conflicting files]
  Automatic merge failed, fix conflicts and then commit the result.
- [nothing]


D
start Git UI
UI started
D
change to working directory
directory changed
D
write git merge [target branch name]
command written
S
give command to execute git merge [target branch name]
To: Git UI + Software (Client)
Msg: git merge [target branch name]
R
waiting for initial git software response
From: Git UI + Software (Client)
Msg: Updating [first commit hash of current branch]..
[last commit hash of current branch]
Fast-forward
[x] file changed, [x] insertions(+)
From: Git UI + Software (Client)
Msg: Merge made by the 'recursive' strategy.
[list of changed files with insertions and deletions]
[x] file changed, [x] insertion(+)
From: Git UI + Software (Client)
Msg: [nothing]
From: Git UI + Software (Client)
Msg: Auto-merging [conflicting files]
CONFLICT (content): Merge conflict in
[list of conflicting files]
Automatic merge failed, fix conflicts
and then commit the result.
D
solve every merge conflict in the list of conflicting files
solved merge conflicts
D
M
git commit the resolved files
resolved files committed
S
give command to execute git commit [resolved files]
To: Git UI + Software (Client)
Msg: git commit [resolved files]
D
continue to work on successfully fast-forward merged branch
D
continue to work on successfully three-way merged branch
D
continue to work on successfully resolved merged branch

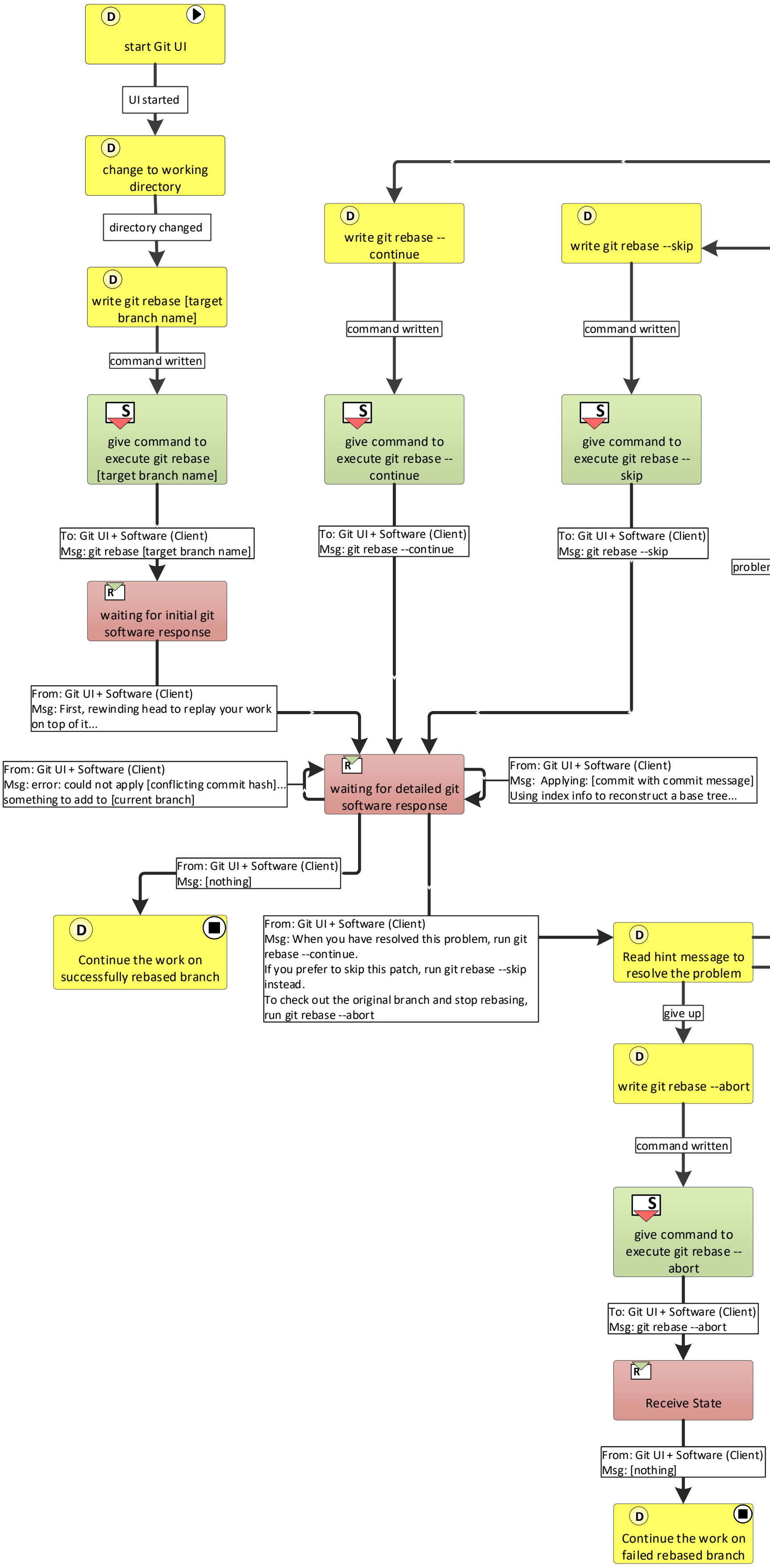

R
Wait for Git User's request
From: Git User
Msg: git merge [target branch name]
D
Check for Diffs between divergent commits
Diffs not solvable via fast-forward or three-way merging
S
send merge conflict notification
To: Git User
Msg: Auto-merging [conflicting files]
CONFLICT (content): Merge conflict in [list of conflicting files]
Automatic merge failed, fix conflicts and then commit the result.
atleast one file solvble via three way merging
No divergent commits available
R
waiting for git user response
From: Git User
Msg: git commit [resolved files]
D
check for ongoing conflicting files
conflictig files still exist
no more conflicts
S
send nothing
To: Git User
Msg: [nothing]
D
Ready
D
set target branch pointer to the latest commit of current branch
target branch pointer set
S
Send notification for fast-forward merge
To: Git User
Msg: Updating [first commit hash of current branch]..
[last commit hash of current branch]
Fast-forward
[x] file changed, [x] insertions(+)
D
Ready
D
merge files three-way recursively
merged
S
Send notification for recursive merge
To: Git User
Msg: Merge made by the 'recursive' strategy.
[list of changed files with insertions and deletions]
[x] file changed, [x] insertion(+)
D
Ready

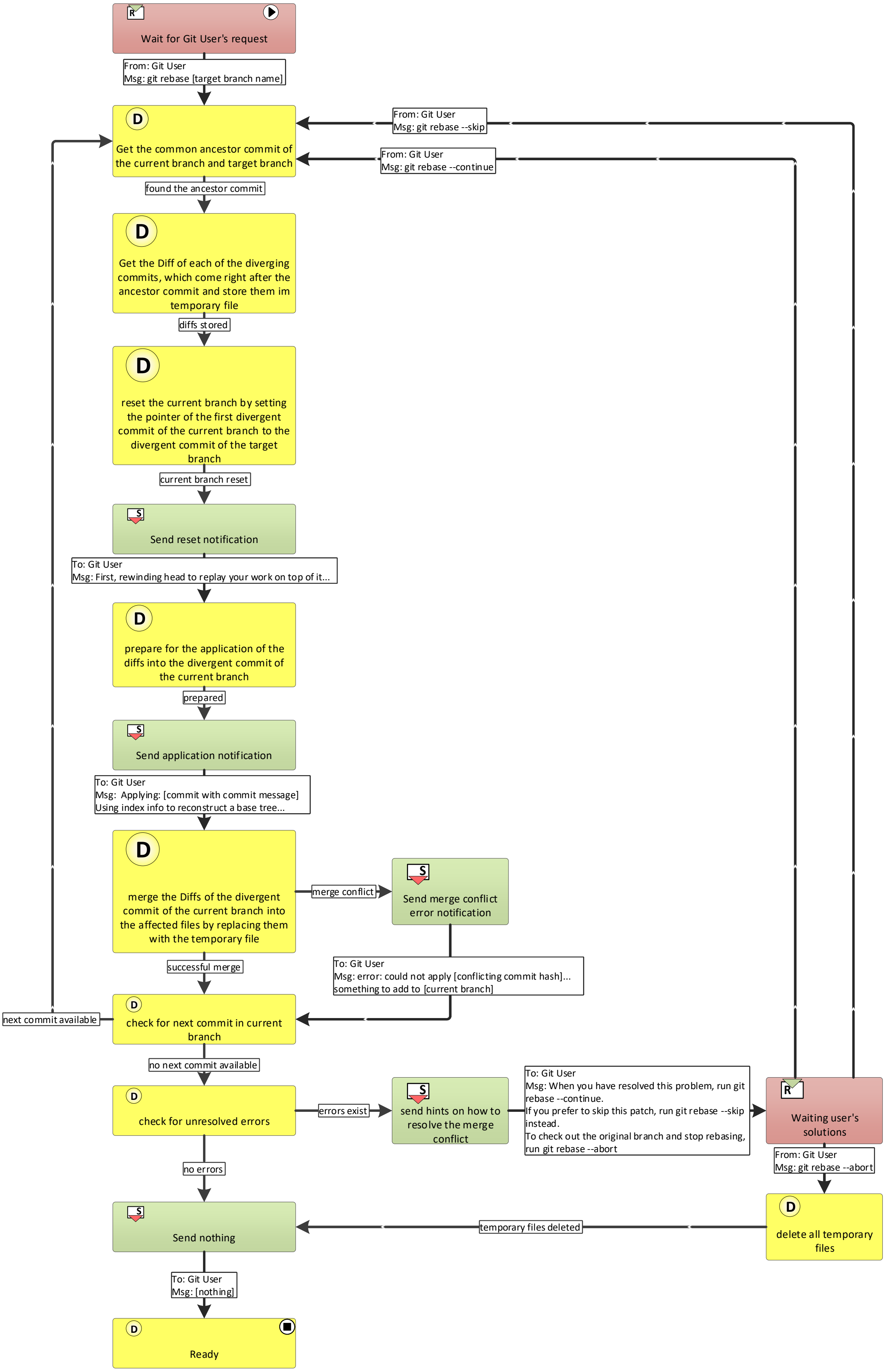
3.4 git rebase



- git rebase [target branch name]
- git rebase --continue
- git rebase --skip
- git rebase --abort

- First, rewinding head to replay your work on top of it...
- Applying: [commit with commit message]
  Using index info to reconstruct a base tree...
- error: could not apply [conflicting commit hash]...
  something to add to [current branch]
- When you have resolved this problem, run git rebase --continue.
  If you prefer to skip this patch, run git rebase --skip instead.
  To check out the original branch and stop rebasing, run git rebase --abort
- [nothing]

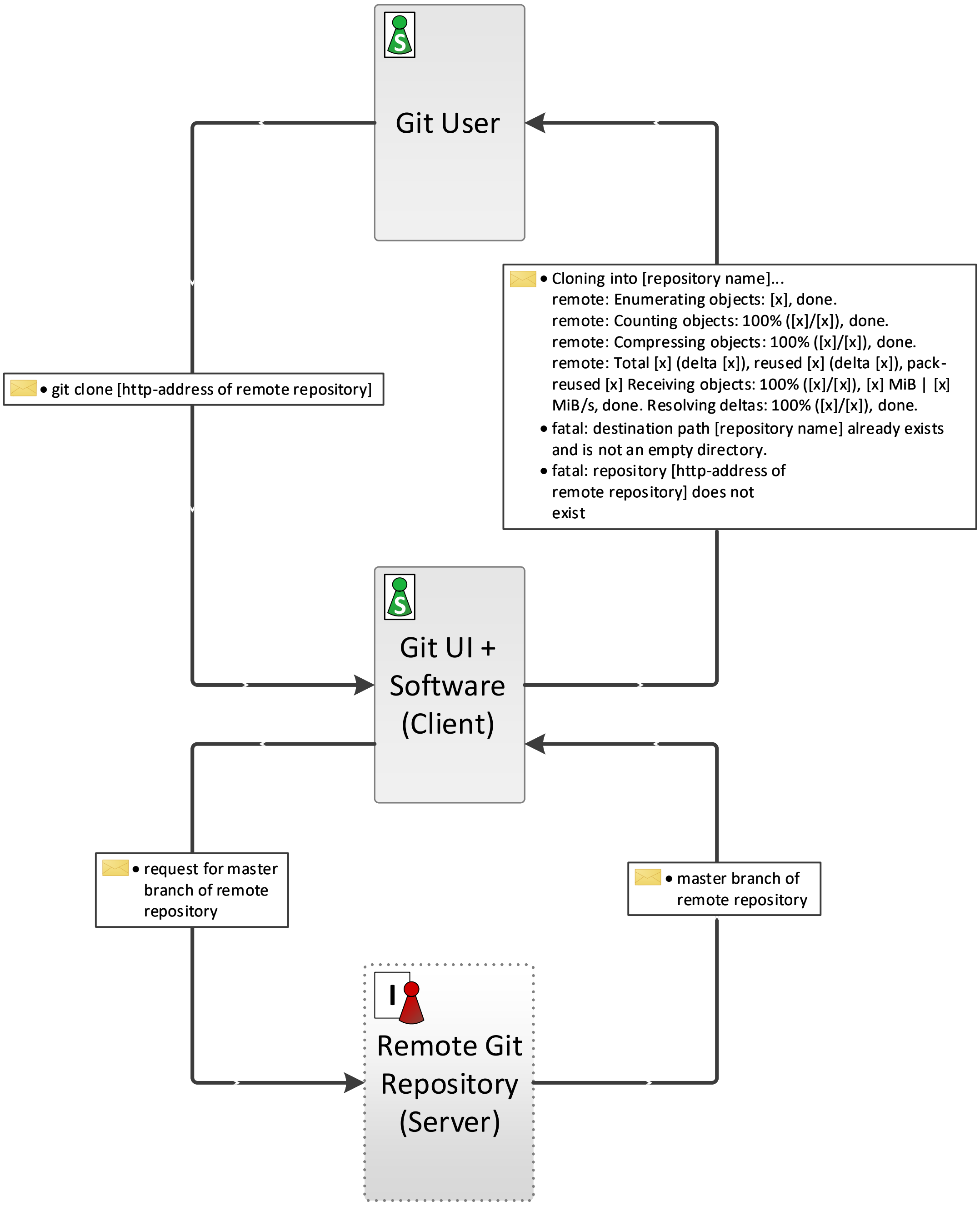

D
start Git UI
UI started
D
change to working directory
directory changed
D
write git rebase [target branch name]
command written
S
give command to execute git rebase [target branch name]
To: Git UI + Software (Client)
Msg: git rebase [target branch name]
R
waiting for initial git software response
From: Git UI + Software (Client)
Msg: First, rewinding head to replay your work on top of it...
D
write git rebase --continue
command written
S
give command to execute git rebase --continue
To: Git UI + Software (Client)
Msg: git rebase --continue
D
write git rebase --skip
command written
S
give command to execute git rebase --skip
To: Git UI + Software (Client)
Msg: git rebase --skip
problem solved
problem skipped
From: Git UI + Software (Client)
Msg: error: could not apply [conflicting commit hash]... something to add to [current branch]
R
waiting for detailed git software response
From: Git UI + Software (Client)
Msg: Applying: [commit with commit message]
Using index info to reconstruct a base tree...
From: Git UI + Software (Client)
Msg: [nothing]
D
Continue the work on successfully rebased branch
From: Git UI + Software (Client)
Msg: When you have resolved this problem, run git rebase --continue.
If you prefer to skip this patch, run git rebase --skip instead.
To check out the original branch and stop rebasing, run git rebase --abort
D
Read hint message to resolve the problem
give up
D
write git rebase --abort
command written
S
give command to execute git rebase --abort
To: Git UI + Software (Client)
Msg: git rebase --abort
R
Receive State
From: Git UI + Software (Client)
Msg: [nothing]
D
Continue the work on failed rebased branch


R
Wait for Git User's request
From: Git User
Msg: git rebase [target branch name]
D
Get the common ancestor commit of the current branch and target branch
From: Git User
Msg: git rebase --skip
From: Git User
Msg: git rebase --continue
found the ancestor commit
D
Get the Diff of each of the diverging commits, which come right after the ancestor commit and store them im temporary file
diffs stored
D
reset the current branch by setting the pointer of the first divergent commit of the current branch to the divergent commit of the target branch
current branch reset
S
Send reset notification
To: Git User
Msg: First, rewinding head to replay your work on top of it...
D
prepare for the application of the diffs into the divergent commit of the current branch
prepared
S
Send application notification
To: Git User
Msg: Applying: [commit with commit message]
Using index info to reconstruct a base tree...
D
merge the Diffs of the divergent commit of the current branch into the affected files by replacing them with the temporary file
merge conflict
S
Send merge conflict error notification
To: Git User
Msg: error: could not apply [conflicting commit hash]...
something to add to [current branch]
successful merge
next commit available
D
check for next commit in current branch
no next commit available
D
check for unresolved errors
errors exist
S
send hints on how to resolve the merge conflict
To: Git User
Msg: When you have resolved this problem, run git rebase --continue.
If you prefer to skip this patch, run git rebase --skip instead.
To check out the original branch and stop rebasing, run git rebase --abort
R
Waiting user's solutions
From: Git User
Msg: git rebase --abort
D
delete all temporary files
temporary files deleted
no errors
S
Send nothing
To: Git User
Msg: [nothing]
D
Ready

4 Git Collaborations

This chapter contains every git command that is related to working on and with remote repositories. Thus, every command in this chapter requires a network connection to work. After finishing this chapter, the reader will be able to clone a remote repository, fetch all the existing branches and commits of that remote repository, pull additional commits from the remote repository and push new commits to the remote repository.

4.1 git clone

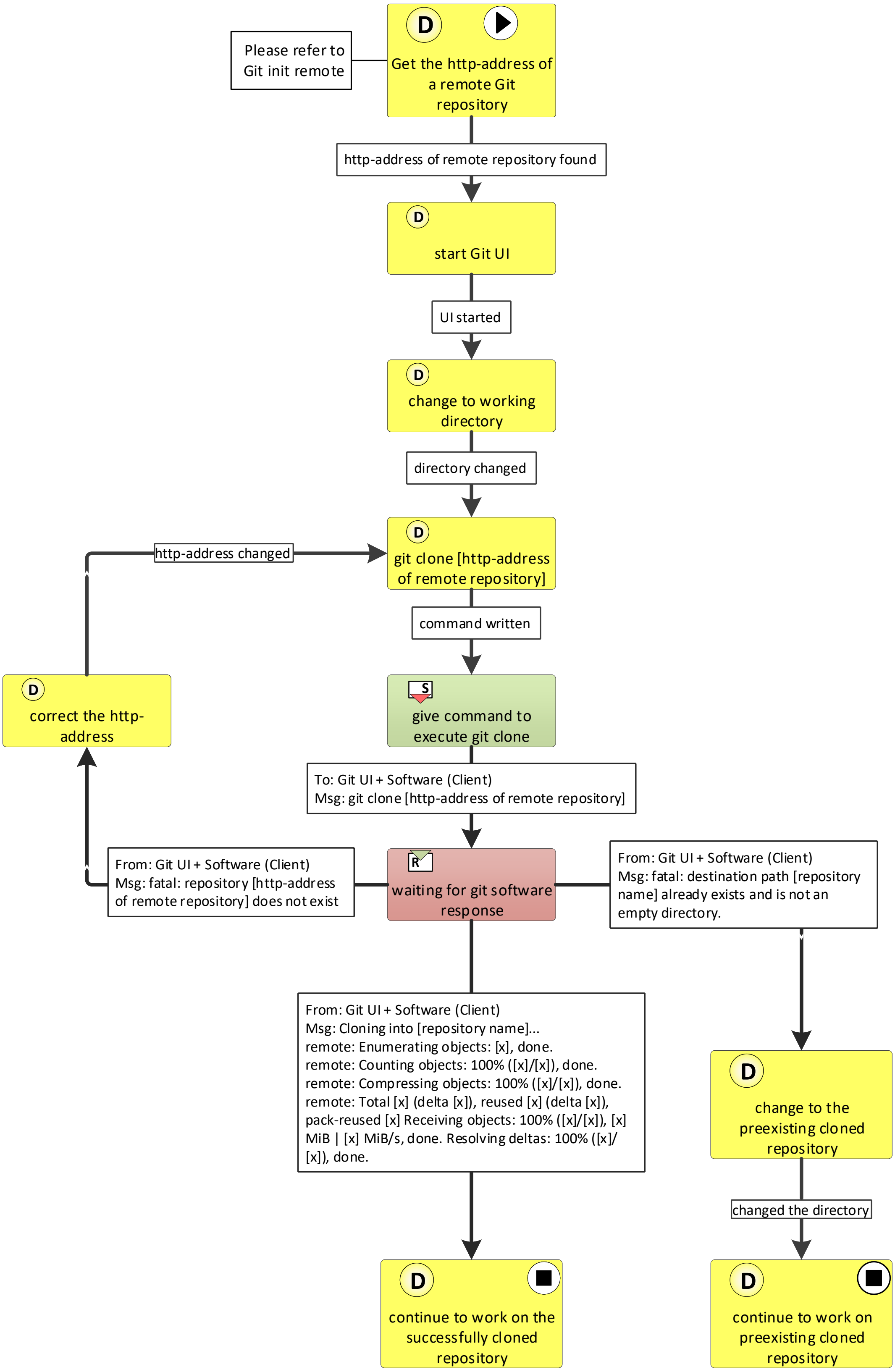


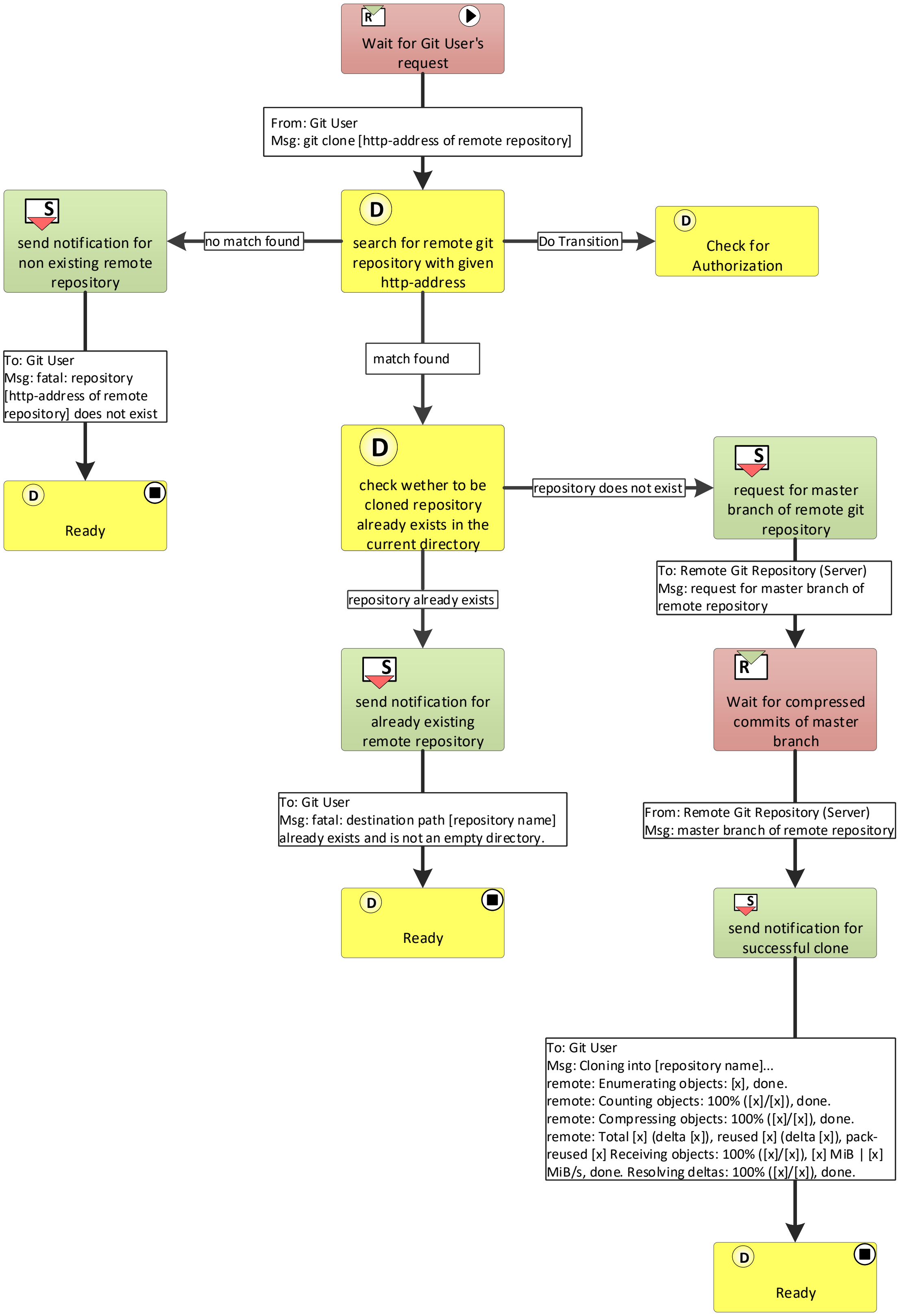
- git clone [http-address of remote repository]

- Cloning into [repository name]... remote: Enumerating objects: [x], done. remote: Counting objects: 100% ([x]/[x]), done. remote: Compressing objects: 100% ([x]/[x]), done. remote: Total [x] (delta [x]), reused [x] (delta [x]), pack-reused [x] Receiving objects: 100% ([x]/[x]), [x] MiB | [x] MiB/s, done. Resolving deltas: 100% ([x]/[x]), done.
- fatal: destination path [repository name] already exists and is not an empty directory.
- fatal: repository [http-address of remote repository] does not exist

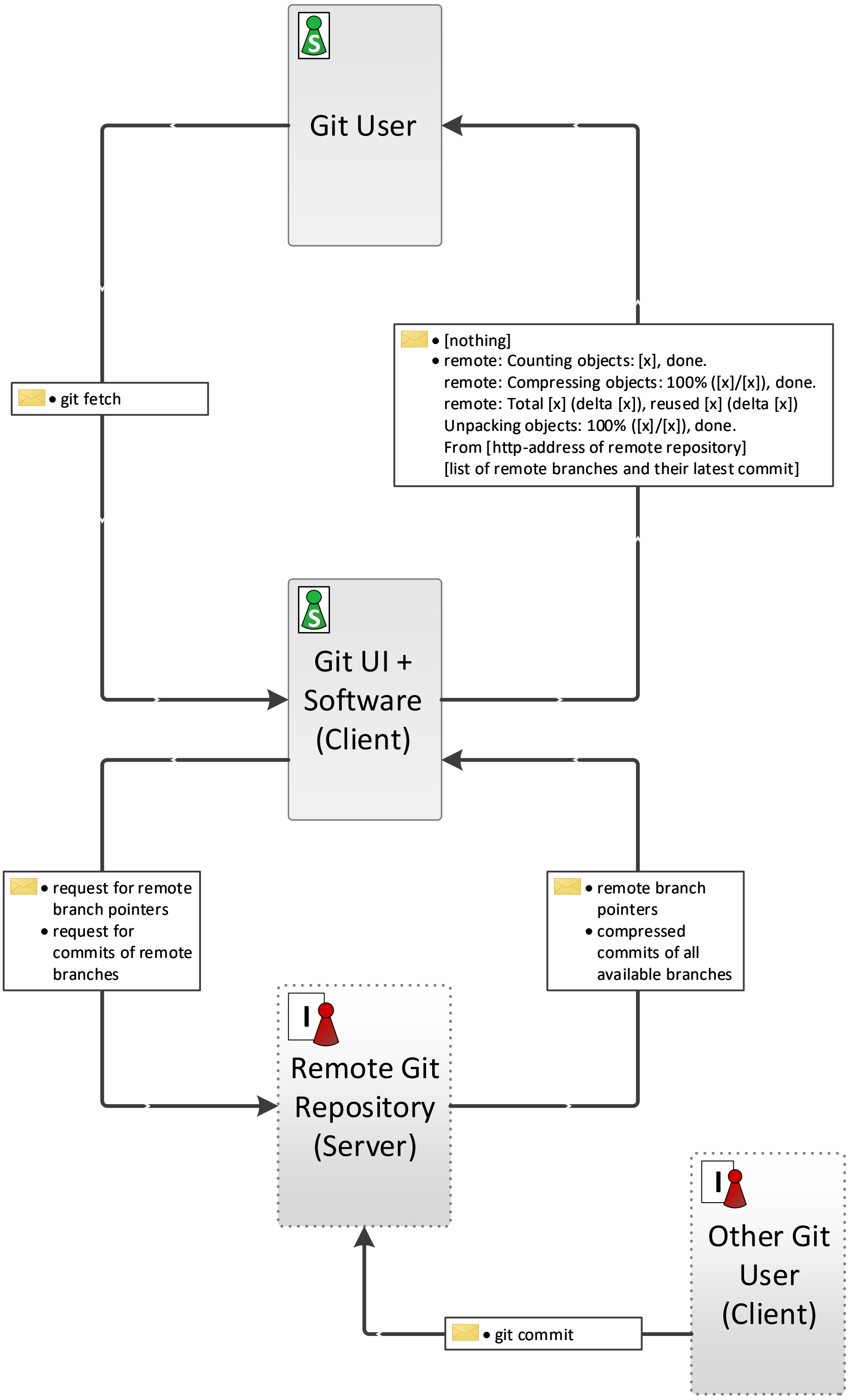
- request for master branch of remote repository

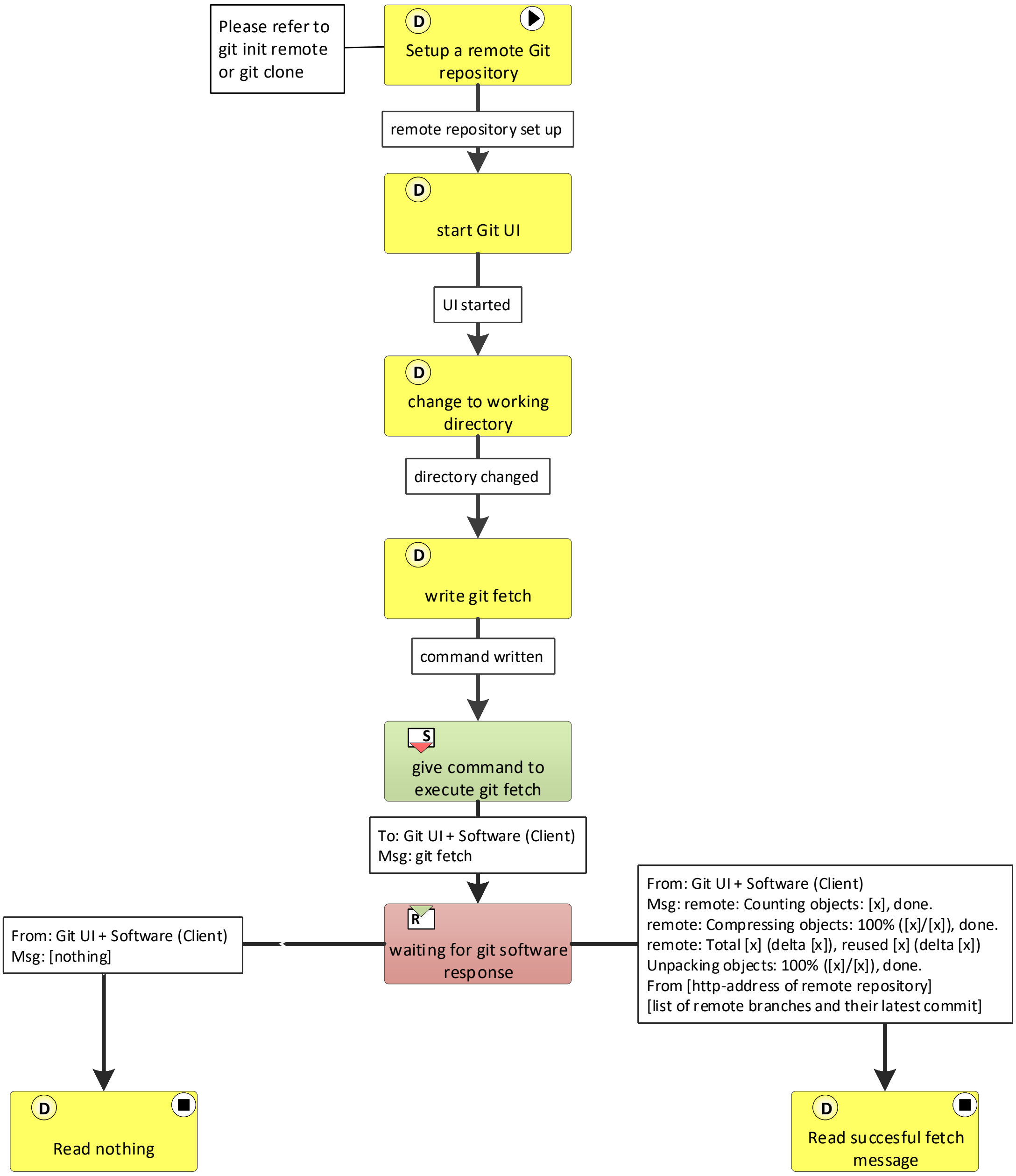
- master branch of remote repository


Please refer to Git init remote
D
Get the http-address of a remote Git repository
http-address of remote repository found
D
start Git UI
UI started
D
change to working directory
directory changed
http-address changed
D
git clone [http-address of remote repository]
command written
D
correct the http-address
S
give command to execute git clone
To: Git UI + Software (Client)
Msg: git clone [http-address of remote repository]
From: Git UI + Software (Client)
Msg: fatal: repository [http-address of remote repository] does not exist
R
waiting for git software response
From: Git UI + Software (Client)
Msg: fatal: destination path [repository name] already exists and is not an empty directory.
From: Git UI + Software (Client)
Msg: Cloning into [repository name]...
remote: Enumerating objects: [x], done.
remote: Counting objects: 100% ([x]/[x]), done.
remote: Compressing objects: 100% ([x]/[x]), done.
remote: Total [x] (delta [x]), reused [x] (delta [x]),
pack-reused [x] Receiving objects: 100% ([x]/[x]), [x]
MiB | [x] MiB/s, done. Resolving deltas: 100% ([x]/
[x]), done.
D
change to the preexisting cloned repository
changed the directory
D
continue to work on the successfully cloned repository
D
continue to work on preexisting cloned repository


R
Wait for Git User's request
From: Git User
Msg: git clone [http-address of remote repository]
D
search for remote git repository with given http-address
no match found
S
send notification for non existing remote repository
To: Git User
Msg: fatal: repository [http-address of remote repository] does not exist
D
Ready
Do Transition
D
Check for Authorization
match found
D
check wether to be cloned repository already exists in the current directory
repository does not exist
S
request for master branch of remote git repository
To: Remote Git Repository (Server)
Msg: request for master branch of remote repository
R
Wait for compressed commits of master branch
From: Remote Git Repository (Server)
Msg: master branch of remote repository
S
send notification for successful clone
To: Git User
Msg: Cloning into [repository name]...
remote: Enumerating objects: [x], done.
remote: Counting objects: 100% ([x]/[x]), done.
remote: Compressing objects: 100% ([x]/[x]), done.
remote: Total [x] (delta [x]), reused [x] (delta [x]), pack-reused [x] Receiving objects: 100% ([x]/[x]), [x] MiB | [x] MiB/s, done. Resolving deltas: 100% ([x]/[x]), done.
D
Ready
repository already exists
S
send notification for already existing remote repository
To: Git User
Msg: fatal: destination path [repository name] already exists and is not an empty directory.
D
Ready

4.2 git fetch

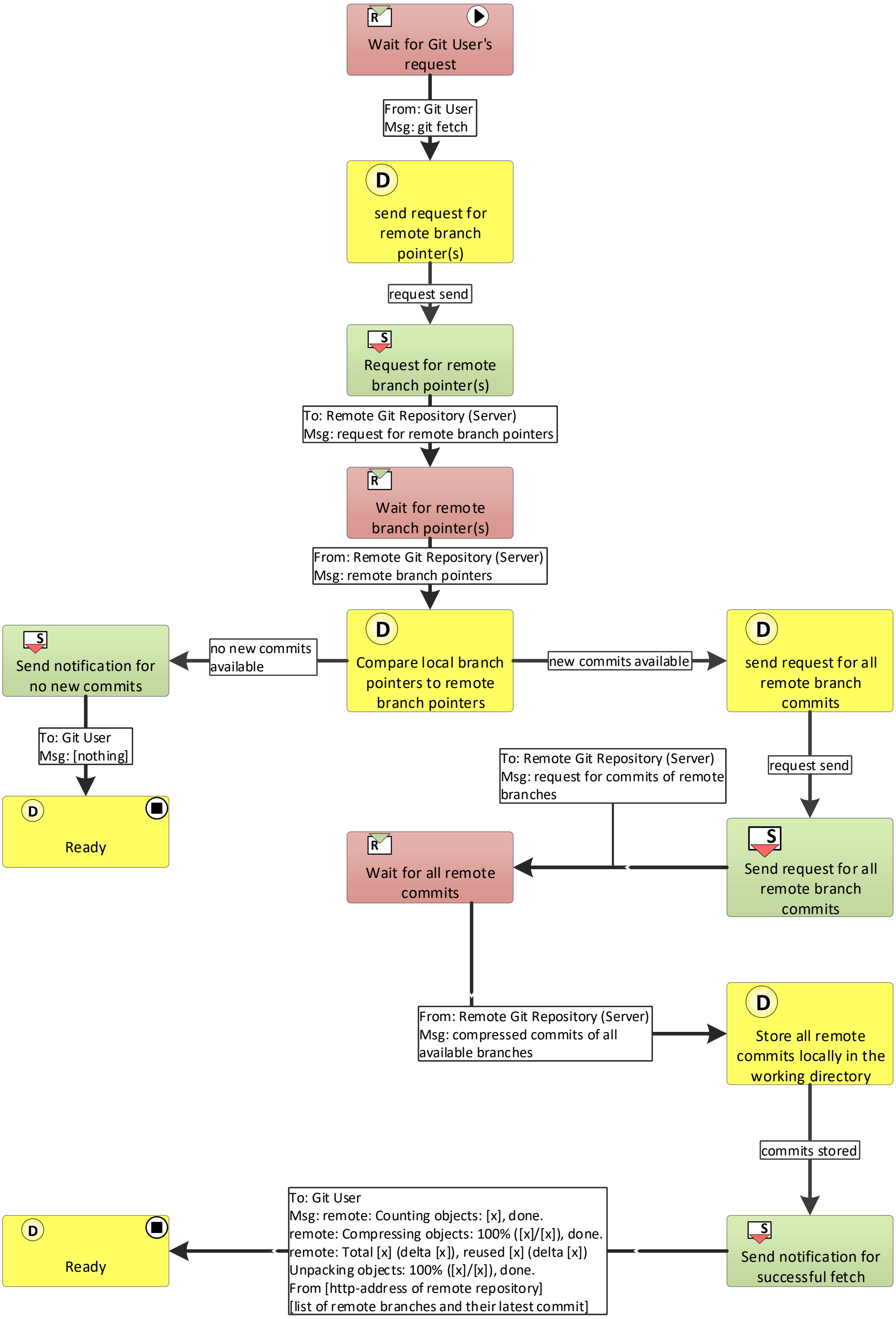

Git User
• git fetch
• [nothing]
• remote: Counting objects: [x], done.
remote: Compressing objects: 100% ([x]/[x]), done.
remote: Total [x] (delta [x]), reused [x] (delta [x])
Unpacking objects: 100% ([x]/[x]), done.
From [http-address of remote repository]
[list of remote branches and their latest commit]
Git UI + Software (Client)
• request for remote branch pointers
• request for commits of remote branches
• remote branch pointers
• compressed commits of all available branches
Remote Git Repository (Server)
• git commit
Other Git User (Client)


Please refer to git init remote or git clone
D
Setup a remote Git repository
remote repository set up
D
start Git UI
UI started
D
change to working directory
directory changed
D
write git fetch
command written
S
give command to execute git fetch
R
waiting for git software response
D
Read nothing
D
Read succesful fetch message

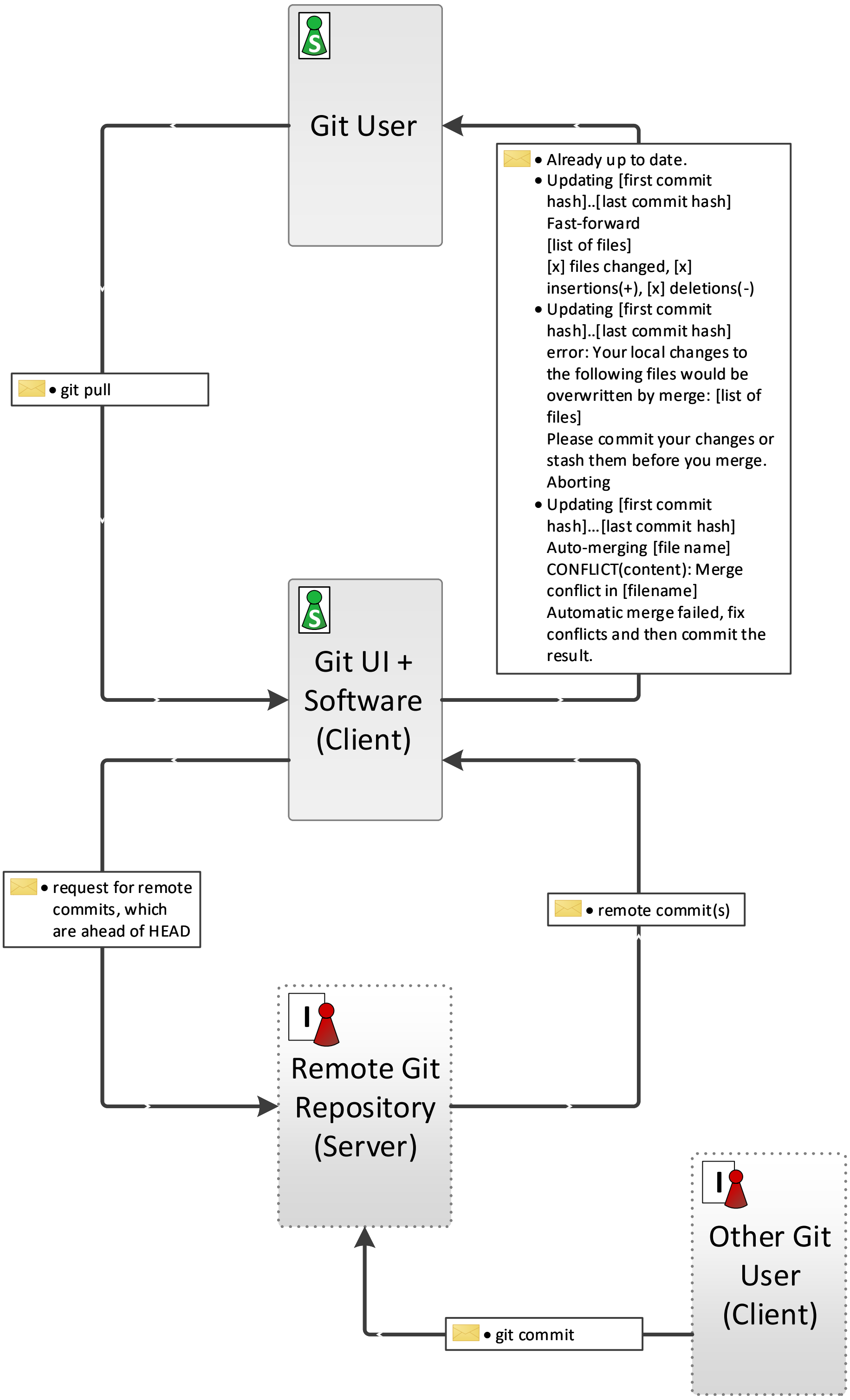
To: Git UI + Software (Client)
Msg: git fetch

From: Git UI + Software (Client)
Msg: [nothing]

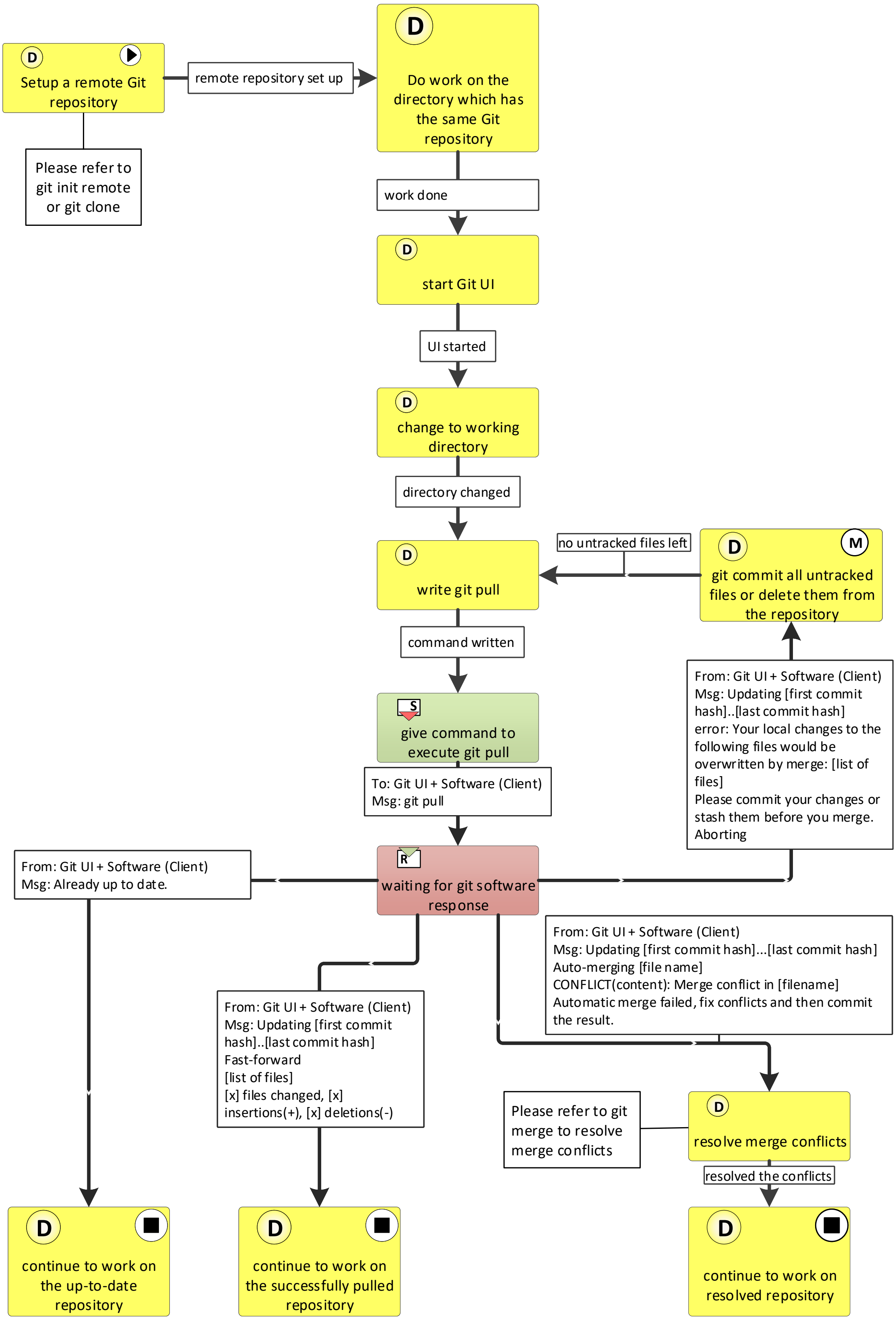
From: Git UI + Software (Client)
Msg: remote: Counting objects: [x], done.
remote: Compressing objects: 100% ([x]/[x]), done.
remote: Total [x] (delta [x]), reused [x] (delta [x])
Unpacking objects: 100% ([x]/[x]), done.
From [http-address of remote repository]
[list of remote branches and their latest commit]


R
Wait for Git User's request
From: Git User
Msg: git fetch
D
send request for remote branch pointer(s)
request send
S
Request for remote branch pointer(s)
To: Remote Git Repository (Server)
Msg: request for remote branch pointers
R
Wait for remote branch pointer(s)
From: Remote Git Repository (Server)
Msg: remote branch pointers
D
Compare local branch pointers to remote branch pointers
no new commits available
S
Send notification for no new commits
To: Git User
Msg: [nothing]
D
Ready
new commits available
D
send request for all remote branch commits
request send
S
Send request for all remote branch commits
To: Remote Git Repository (Server)
Msg: request for commits of remote branches
R
Wait for all remote commits
From: Remote Git Repository (Server)
Msg: compressed commits of all available branches
D
Store all remote commits locally in the working directory
commits stored
S
Send notification for successful fetch
To: Git User
Msg: remote: Counting objects: [x], done.
remote: Compressing objects: 100% ([x]/[x]), done.
remote: Total [x] (delta [x]), reused [x] (delta [x])
Unpacking objects: 100% ([x]/[x]), done.
From [http-address of remote repository]
[list of remote branches and their latest commit]
D
Ready

4.3 git pull


Git User
• git pull
Git UI + Software (Client)
• request for remote commits, which are ahead of HEAD
• remote commit(s)
Remote Git Repository (Server)
Other Git User (Client)
• git commit

- Already up to date.
- Updating [first commit hash]..[last commit hash]
  Fast-forward
  [list of files]
  [x] files changed, [x] insertions(+), [x] deletions(-)
- Updating [first commit hash]..[last commit hash]
  error: Your local changes to the following files would be overwritten by merge: [list of files]
  Please commit your changes or stash them before you merge.
  Aborting
- Updating [first commit hash]...[last commit hash]
  Auto-merging [file name]
  CONFLICT(content): Merge conflict in [filename]
  Automatic merge failed, fix conflicts and then commit the result.


D
Setup a remote Git repository
Please refer to git init remote or git clone
remote repository set up
D
Do work on the directory which has the same Git repository
work done
D
start Git UI
UI started
D
change to working directory
directory changed
D
write git pull
no untracked files left
D
M
git commit all untracked files or delete them from the repository
command written
S
give command to execute git pull
R
waiting for git software response
D
resolve merge conflicts
Please refer to git merge to resolve merge conflicts
resolved the conflicts
D
continue to work on the up-to-date repository
D
continue to work on the successfully pulled repository
D
continue to work on resolved repository

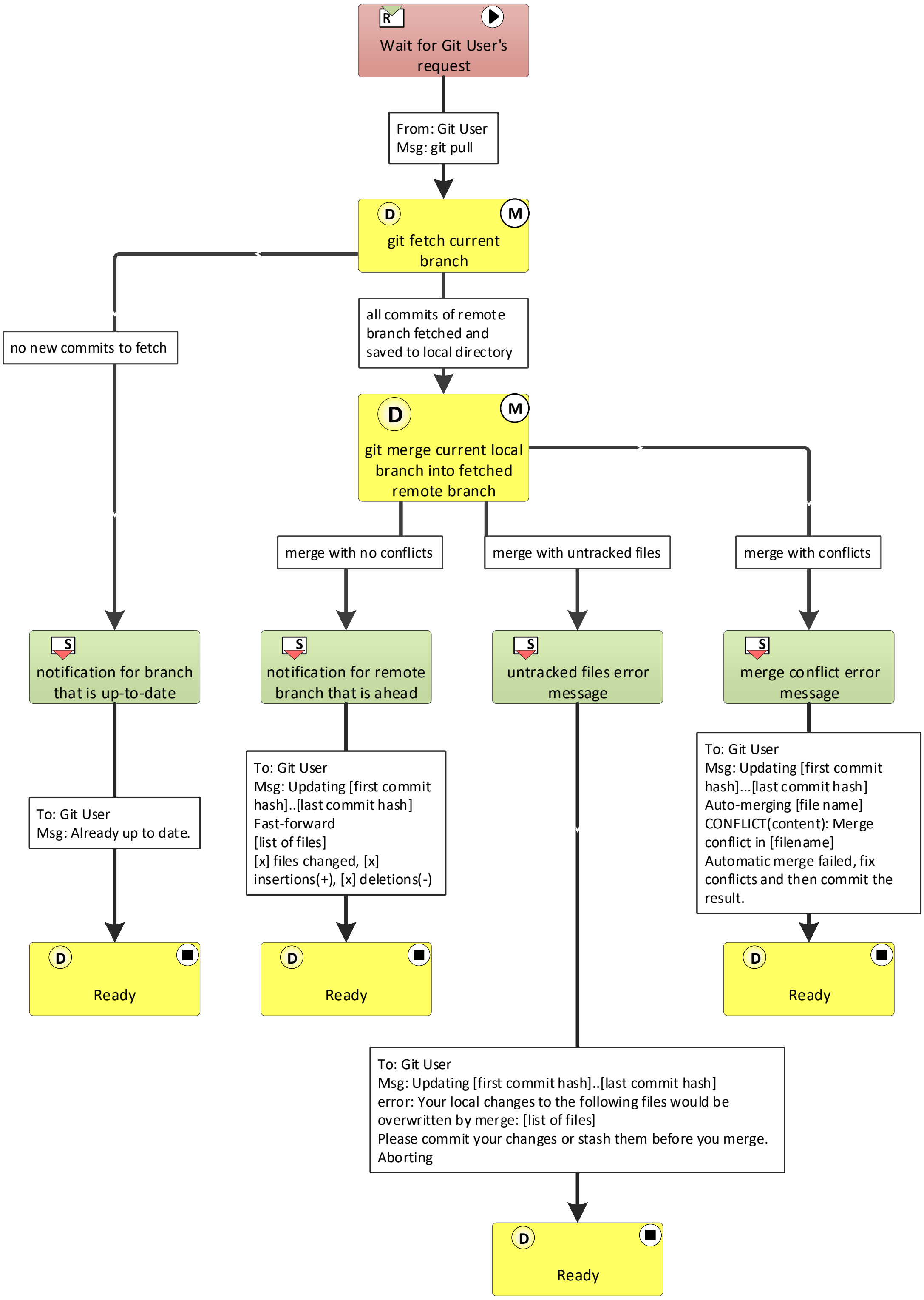
To: Git UI + Software (Client)
Msg: git pull

From: Git UI + Software (Client)
Msg: Updating [first commit hash]..[last commit hash]
error: Your local changes to the following files would be overwritten by merge: [list of files]
Please commit your changes or stash them before you merge.
Aborting

From: Git UI + Software (Client)
Msg: Already up to date.

From: Git UI + Software (Client)
Msg: Updating [first commit hash]..[last commit hash]
Fast-forward
[list of files]
[x] files changed, [x] insertions(+), [x] deletions(-)

From: Git UI + Software (Client)
Msg: Updating [first commit hash]...[last commit hash]
Auto-merging [file name]
CONFLICT(content): Merge conflict in [filename]
Automatic merge failed, fix conflicts and then commit the result.


R
Wait for Git User's request
D
M
git fetch current branch
no new commits to fetch
all commits of remote branch fetched and saved to local directory
D
M
git merge current local branch into fetched remote branch
merge with no conflicts
merge with untracked files
merge with conflicts
S
notification for branch that is up-to-date
S
notification for remote branch that is ahead
S
untracked files error message
S
merge conflict error message
D
Ready
D
Ready
D
Ready
D
Ready

From: Git User
Msg: git pull

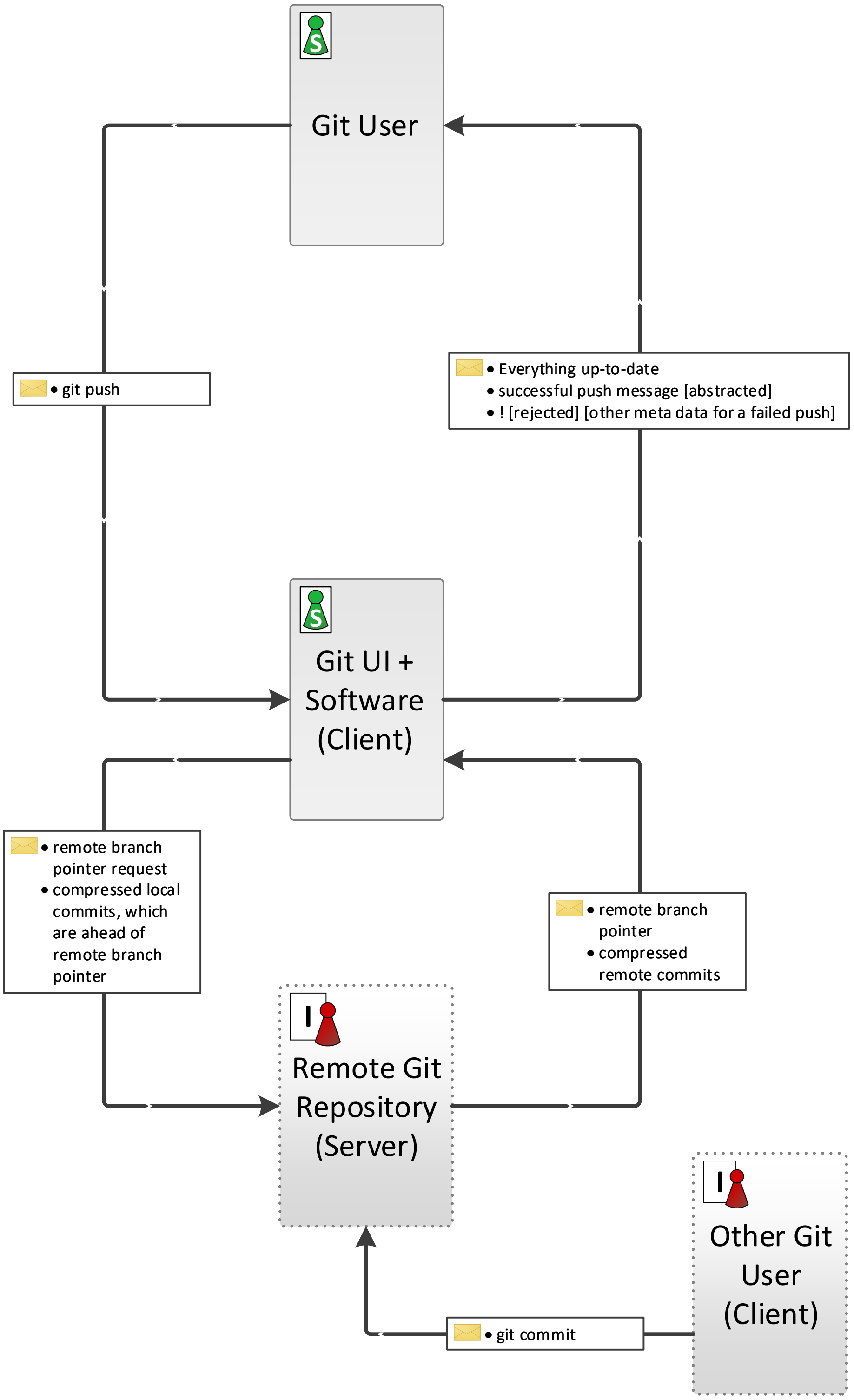
To: Git User
Msg: Already up to date.

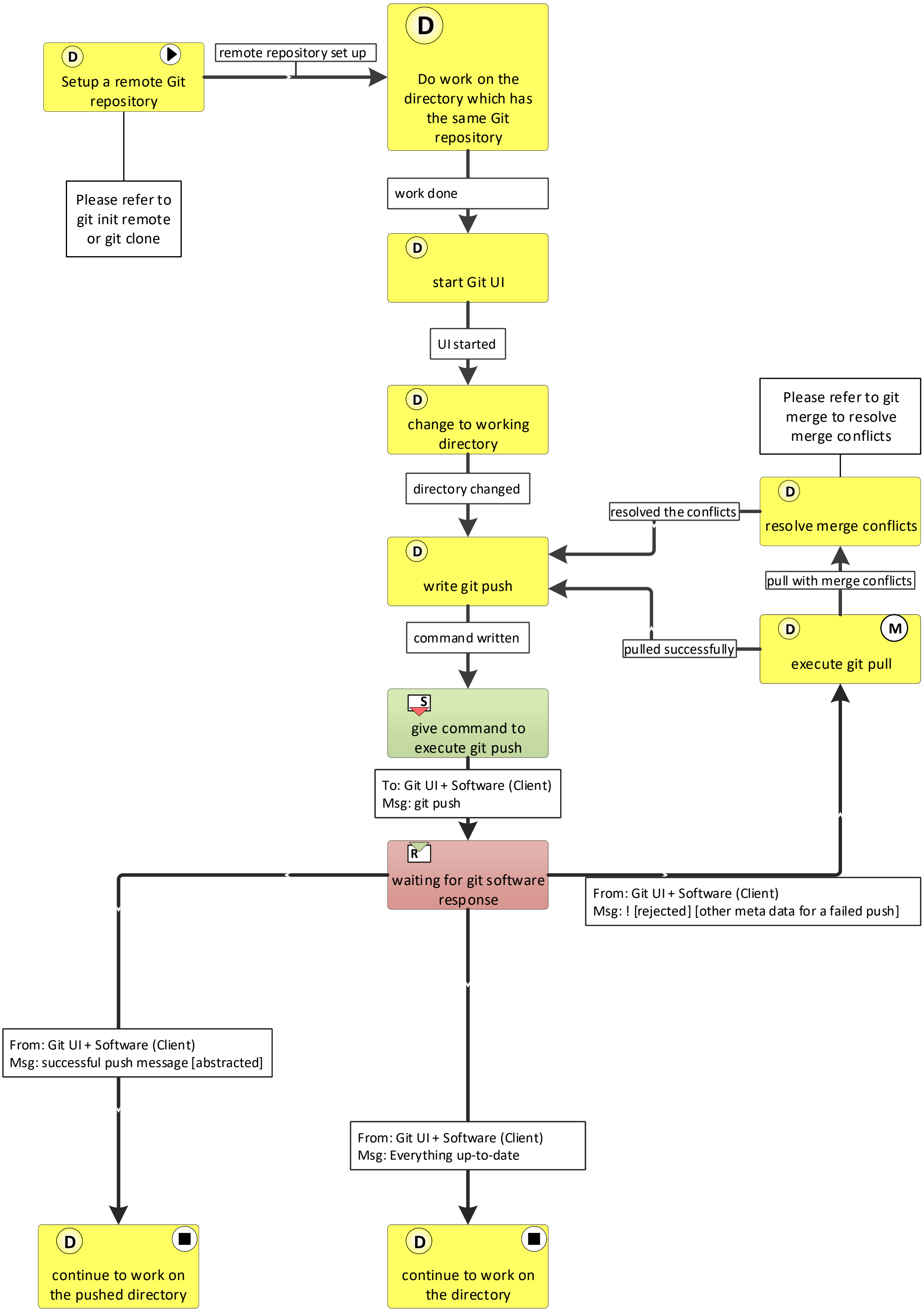
To: Git User
Msg: Updating [first commit hash]..[last commit hash]
Fast-forward
[list of files]
[x] files changed, [x] insertions(+), [x] deletions(-)

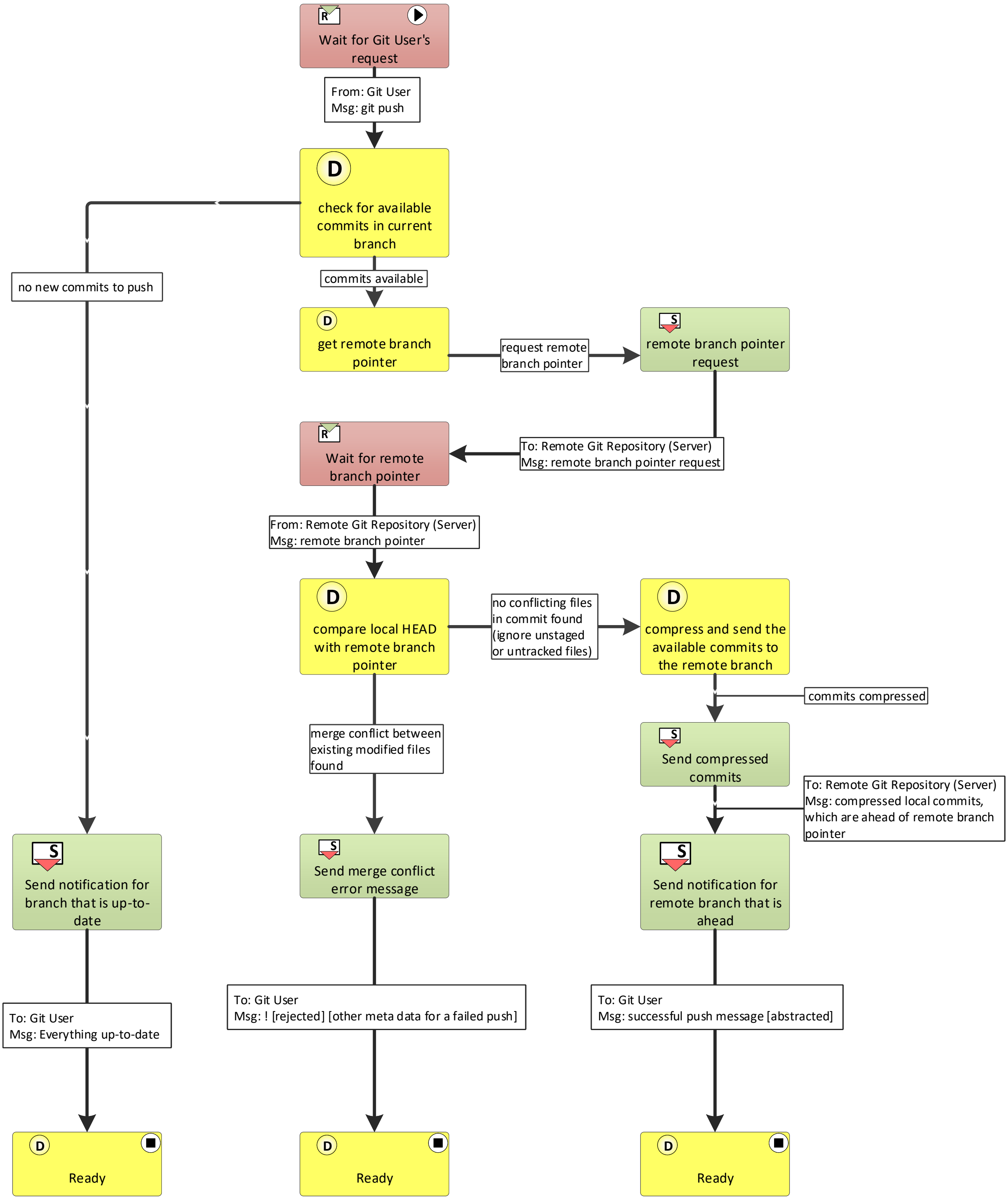
To: Git User
Msg: Updating [first commit hash]...[last commit hash]
Auto-merging [file name]
CONFLICT(content): Merge conflict in [filename]
Automatic merge failed, fix conflicts and then commit the result.

To: Git User
Msg: Updating [first commit hash]..[last commit hash]
error: Your local changes to the following files would be overwritten by merge: [list of files]
Please commit your changes or stash them before you merge.
Aborting

4.4 git push


Git User
• git push
• Everything up-to-date
• successful push message [abstracted]
• ! [rejected] [other meta data for a failed push]
Git UI + Software (Client)
• remote branch pointer request
• compressed local commits, which are ahead of remote branch pointer
• remote branch pointer
• compressed remote commits
Remote Git Repository (Server)
• git commit
Other Git User (Client)


D
Setup a remote Git repository
remote repository set up
D
Do work on the directory which has the same Git repository
Please refer to git init remote or git clone
work done
D
start Git UI
UI started
D
change to working directory
Please refer to git merge to resolve merge conflicts
directory changed
resolved the conflicts
D
resolve merge conflicts
D
write git push
pull with merge conflicts
D
M
execute git pull
command written
pulled successfully
S
give command to execute git push
To: Git UI + Software (Client)
Msg: git push
R
waiting for git software response
From: Git UI + Software (Client)
Msg: ! [rejected] [other meta data for a failed push]
From: Git UI + Software (Client)
Msg: successful push message [abstracted]
From: Git UI + Software (Client)
Msg: Everything up-to-date
D
continue to work on the pushed directory
D
continue to work on the directory


R
Wait for Git User's request
From: Git User
Msg: git push
D
check for available commits in current branch
no new commits to push
commits available
D
get remote branch pointer
request remote branch pointer
S
remote branch pointer request
R
Wait for remote branch pointer
To: Remote Git Repository (Server)
Msg: remote branch pointer request
From: Remote Git Repository (Server)
Msg: remote branch pointer
D
compare local HEAD with remote branch pointer
no conflicting files in commit found (ignore unstaged or untracked files)
D
compress and send the available commits to the remote branch
commits compressed
merge conflict between existing modified files found
S
Send compressed commits
To: Remote Git Repository (Server)
Msg: compressed local commits, which are ahead of remote branch pointer
S
Send notification for branch that is up-to-date
S
Send merge conflict error message
S
Send notification for remote branch that is ahead
To: Git User
Msg: Everything up-to-date
To: Git User
Msg: ! [rejected] [other meta data for a failed push]
To: Git User
Msg: successful push message [abstracted]
D
Ready
D
Ready
D
Ready

5 Appendix

6 Bibliography

[LM12] Jon Loeliger and Matthew McCullough: Version Control with Git, OŔEILLY, ISBN ISBN 978-1-449-31638-9, 2012.

[El21] Matthes Elstermann: Thoughts and Concepts on Subject Oriented Business Process Management & Modelling, `https://subjective-me.jimdofree.com/visio-modelling/`, 2021.

[CS20] Scott Chacon and Ben Straub: Pro Git EVERYTHING YOU NEED TO KNOW ABOUT GIT, `https://git-scm.com/book/en/v2`, 2020.